Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Håkon Flisnes Johansen

Development of prototype for remote commissioning and service of fire safety equipment

Master's thesis in Cybernetics and Robotics
Supervisor: Mary Ann Lundteigen
January 2021

NTNU
Norwegian University of
Science and Technology

Håkon Flisnes Johansen

Development of prototype for remote commissioning and service of fire safety equipment

Master's thesis in Cybernetics and Robotics
Supervisor: Mary Ann Lundteigen
January 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

The goal of this master's thesis was to determine if the AutroLink prototype could be further developed into a tool to aid in Autronica Fire and Security AS' service and commissioning procedures of their AutroSafe 4 fire detection system. The thesis is a continuation of a specialization project were the AutroLink prototype was designed, implemented and tested. This prototype would monitor the internal bus communication in the AutroSafe 4 fire detection system, and transmitted the information to an Internet-connected server.

Information about the AutroSafe system, as well as Autronica's tools and routines for service and commissioning, was gathered in a system description. A literature study on subjects such as remote access technologies, software development and real-time systems was conducted. Information from the literature study and system description was analyzed in order to locate functionality that would benefit Autronica's service and commissioning routines. Here it was determined that the ability to bring in expert help remotely, as well as the opportunity to run certain procedures automatically would be beneficial. The most fitting target was the AutroSafe system's module stack and detection loops. In order to facilitate this, it was necessary to introduce two-way communication. New requirements were therefore defined for the prototype. Furthermore, a structured analysis was performed, modularizing the functionalities of the prototype, culminating in a new specification. The prototype was modified according to the new specification.

The AutroLink prototype's new functionalities were validated through testing. In conclusion, we were able to verify that the prototype could be a useful service and commissioning tool for the AutroSafe 4 fire detection system since it was able to facilitate remote service and could perform certain procedures automatically.

Sammendrag

Målet med denne masteroppgaven var å avgjøre om AutroLink-prototypen kunne videreutvikles til et verktøy til bruk under Autronica Fire and Security AS' service- og igangkjøringsprosedyrer på AutroSafe 4 branndeteksjonssystemet. Oppgaven er en videreføring av et spesialiseringsprosjekt der AutroLink-prototypen ble designet, implementert og testet. Denne prototypen kan overvåke den interne busskommunikasjonen i AutroSafe 4 branndeteksjonssystemet og overføre informasjonen til en Internett-tilkoblet server.

Informasjon om AutroSafe systemet, samt Autronicas verktøy og rutiner for service og igangkjøring, ble samlet i en systembeskrivelse. En litteraturstudie om ekstern tilgangsteknologi, programvareutvikling og sanntidssystemer ble gjennomført. Informasjon fra litteraturstudien og systembeskrivelsen ble analysert for å finne funksjonalitet som ville være til nytte for Autronicas service- og idriftsettingsrutiner. Her ble det bestemt at evnen til å hente inn eksperthjelp eksternt, samt muligheten til å kjøre visse prosedyrer automatisk ville være nyttig. Det ble valgt å koble prototypen mot AutroSafe-systemets modulstakk og detektorsløyfer. Dette gjorde at det var nødvendig å innføre toveiskommunikasjon. Nye krav ble derfor definert for prototypen. Videre ble det utført en strukturert analyse som modulariserte funksjonene til prototypen, som endte i en ny spesifikasjon. Prototypen ble modifisert i henhold til den nye spesifikasjonen.

AutroLink-prototypens nye funksjoner ble validert gjennom testing. Det ble konkludert med at prototypen var et nyttig service- og igangkjøringsverktøy for AutroSafe 4 branndeteksjonssystem, siden den tilrettela for fjernservice og kunne utføre visse prosedyrer automatisk.

Preface

This thesis is titled "Development of prototype for remote commissioning and service of fire safety equipment", and is written as part of the course TTK4900, Engineering Cybernetics, Master's Thesis, at the Norwegian University of Science and Technology. The course is weighted 30 ECTS credits. The thesis is a continuation of the course TTK4551 Engineering Cybernetics, Specialization Project which was completed during the spring semester of 2020.

I would like to thank Autronica F&S for the opportunity to work on this project. They provided information on their products and access to their internal documentation, which was of great help in writing this thesis.

I would also like to thank my supervisors Martin Langås from Autronica and Prof. Mary Ann Lundteigen from NTNU. They were both of great help, and provided valuable guidance throughout the project period. I would also like to thank Martin for the help with the testing of the prototype.

I would like to thank Nordic Semiconductor for providing me with a free nRF9160 development kit, and for providing invaluable support for their devices through the forums at the Nordic Devzone. The MQTT event handler that is part of the firmware written for the prototype is based on the simple_MQTT sample from Nordic Semiconductor. This sample can be found in the nRF Connect SDK.

Lastly I'd like to thank my friends, family and partner, who provide me with constant support in all I do.

Trondheim January 27, 2021

Contents

Glossary

AL_Com Autronica's proprietary communication protocol used between the loop driver (BSD-310) and loop units. 10, 12, 40, 43, 47, 50, 52, 55, 56, 58, 61, 64, 71, 75, 78, 84

AL_Com+ Autronica's proprietary communication protocol used between a controller and a module stack. 7, 8, 12, 17, 21, 23, 40, 43–48, 50–53, 55, 56, 58–61, 63–65, 71, 72, 75, 79, 84

Aperiodic A task that may appear at any time in a real-time system. 31, 45, 55

AutroSafe Fire alarm system from Autronica, described in section (ref). 5, 14, 16, 45–47

Baud A measure of how many times per second symbols can change on a communication channel. Baud rate is also known as symbol rate. 10

Detection loop The detection loop, also called "detector loop" or "AL_Com loop", is a series of interconnected fire/smoke/gas detecting equipment where both ends of the wire is connected to a loop driver, forming a loop of detection units. 5, 6, 9, 40, 41, 43, 74, 75

Firmware Permanent software programmed into a read-only memory. 51

Hardware The machines, wiring, and other physical components of a computer or other electronic system. 50

Loop unit Loop unit is an umbrella-term for the Autronica devices that connect to the detector loop. E.g. fire, smoke and gas detectors. 8, 9, 40, 41, 75

Module stack The module stack is a collection of devices that connect with control panels and controllers to expand their functionalities. 5, 8, 40–44, 47, 55, 57, 58, 60, 74, 75

Periodic A task that appears at known intervals in a real-time system. 31

RS-232 Recommended Standard 232 is a standard that defines electrical characteristics and timing for a type of serial data transmission. 22, 44, 71

Sporadic A task that may appear at any time in a real-time system, but there is a minimum time between them. 31

Wi-Fi Wi-Fi is a trademarked phrase used about a family of wireless networking technologies based on the IEEE 802.11 family of standards. 37

Acronyms

AR Augmented Reality. 25

ASCII American Standard Code for Information Interchange. 53, 59, 60, 84

CTS Clear To Send. 13, 50, 62

DK Development Kit. 36, 49

FIFO First-in First-out. 53, 54, 56, 64–66

GPIO General-Purpose Input/Output. 22, 36, 50, 62, 63, 78

HAZOP Hazard and Operability study. 27

I2C Inter-Integrated Circuit. 36

IACS Industrial Automation and Control System. 27

IoT Internet of Things. 28, 36, 37, 78

IP Internet Protocol. 28

ISR Interrupt Service Routine. 62, 64, 65

LED Light Emitting Diode. 8, 16, 39, 74, 75

LPWAN Low Power Wide Area Network. 50

LTE-M Long-Term Evolution Machine Type Communication. 21, 36, 50, 78, 84

M2M Machine to Machine. 28

MCU Microcontroller Unit. 37, 38

MQTT Message Queuing Telemetry Transport. 23, 28–30, 37, 40, 44–48, 50, 53

NB-IoT Narrowband Internet of Things. 21, 36, 84

PCB Printed Circuit Board. xv, 22, 37, 84

QoS Quality of Service. 30, 53

RTOS Real-Time Operating System. 37, 38

RTS Request To Send. 13, 17, 50, 58, 62, 74, 78

RX Receive. 22

SDK Software Development Kit. 49

SIL Safety Integrity Level. 26, 27

SiP System in Package. 36, 53

SIS Safety Instrumented System. 26, 27

SoC System on Chip. 36

SPI Serial Peripheral Interface. 36

TCP Transmission Control Protocol. 28

TTL Transistor-Transistor Logic. 22, 44

TX Transmit. 22, 63

UART Universal Asynchronous Receiver/Transmitter. 36, 47, 48, 56, 62, 64, 71, 78

USB Universal Serial Bus. 71

List of Figures

Chapter 1

Introduction

1.1 Background and motivation

Autronica Fire and Security, from here on out referred to as Autronica, is a developer and manufacturer of fire-safety solutions. They deliver complete systems to land, petrochemical oil and gas installations both onshore and offshore, as well as systems for marine vessels.

During the spring semester of 2020, a report regarding wireless data-collection from the AutroSafe 4 fire alarm system was written [1]. The data in question were the AL_ Com+ messages passed between the loop drivers and controllers in the system. These messages could further be analyzed to gain insight into the state of the system.

A prototype based on Nordic Semiconductor's nRF91 SiP was designed and constructed. The prototype monitored and captured the serial data exchanged between the loop drivers and panels. This information was then sorted, filtered and sent to a Internet-connected server via the LTE-M network. Additional clients could connect to the MQTT broker through the Internet in order to receive the stream of AL_ Com+ messages. The prototype was named AutroLink.

The prototype developed during the specialization project showed a lot of promise in other use-cases. One of these were during commissioning and service of Autronica's fire safety systems. Access to the lower parts of the system, such as the detectors, module stack, and insight into the communication between them could prove useful. In the specialization project, re-purposing the AutroLink prototype as a tool that could be used during commissioning was suggested as future work.

Since the new prototype will be based on the AutroLink prototype from the project report, the prototype developed in this master's thesis will be referred to as AutroLink v2, while the original prototype will be referred to as AutroLink v1.

1.2 Goals and tasks

The goal of this master's thesis is to determine if a prototype, such as the AutroLink v1, can be used as a tool to aid Autronica's service and commissioning procedures. This entails further developing the AutroLink v1 prototype as a tool, with added functionality that can aid during the service and commissioning phases of Autronica's AutroSafe fire alarm system. In order to achieve this, the following list of tasks were defined as part of the problem description.

T1: Familiarize with and describe Autronica's:

- Systems and components that are of relevance to this thesis
- Commissioning and service routines

T2: Conduct and document a literature study of relevant technologies and concepts. In addition, examine to what extent commissioning and service with remote access has been discussed in literature

T3: Based on the information from the system description and literature study; propose a design for a prototype, including a structured analysis and specifications

T4: Implement and test the suggested design in a prototype

T5: Discuss the results against the specification

T6: Identify and suggest possible future work and opportunities

1.3 Research approach

In order to gain an understanding of the AutroSafe system, publicly available documents pertaining the system will be studied. Questions regarding the systems, especially the finer details, will be discussed with the engineers of Autronica. They will also likely be the source of the explanations of the service and commissioning routines. Beyond this, internal documents detailing the system will be requested in order to gain a greater understanding. The report written as part of the specialization project also contains information that could be relevant in this thesis.

When conducting the literature study, the focus will be to locate relevant documents, articles and other reputable publications. Search engines such as NTNU Oria and Google Scholar will be the primary targets for inquiry. Search terms such as "remote access", "industrial safety system", "legacy system", "service", "commissioning", "fire detection" and "gas detection" will be used. The literature study will likely also contain information on other subjects that will be of relevance, such as the technologies used in the design, implementation and testing of the prototype.

When working with the development of the new prototype, I want to employ an existing, well documented and tested work method. This could be beneficial as it could streamline the process. Especially when considering that it is highly likely that new code will be written for the new prototype. It was chosen to take inspiration from the waterfall and the v-model methods. This can be seen as the subjects for the chapters four through six reflect the different phases of these models.

1.4 Limitations

Information regarding the internal procedures and practices of Autronica's equipment will be kept to a minimum. Sensitive information will be withheld. In the bibliography, certain references are marked as "Autronica [Internal document]". These documents are not available to the public.

Performing a cybersecurity analysis of the prototype and the AutroSafe system would be beneficial as the introduction of remote access exposes the system to potential cybersecurity risks. However, there are two reasons this is not within the scope of this thesis. The first being that cybersecurity analyses represents a large field of study, including it in the thesis would be too extensive. The second reason being that Autronica requires a legal non-disclosure agreement in order to share this information.

As this thesis and the problem description is a continuation of the specialization project, the existing prototype and hardware will be reused to the extent that it allows. Implementations, designs or solutions using other equipment than the previous prototype will not be covered in this thesis.

Another limitation in this project is that the prototype should function without having to make any software, firmware or hardware changes to the existing AutroSafe equipment. This will ensure that the prototype is compatible with any existing AutroSafe 4 installation.

The Covid-19 pandemic lead to some limitations on the project. This mainly manifested in the form of limited access to equipment and Autronica's documentation. It also lead to there being no large scale testing of the prototype, in order to limit the spread of infection.

1.5 Contributions

This thesis contributes with a prototype gateway that connects with the AutroSafe system, and allows for service engineers to communicate wirelessly through the cellular LTE-M network with AutroSafe equipment.

Parts of the system description in Chapter 2 was taken from the literature study conducted as part of the specialization project preceding this thesis. These parts were edited and revised to better fit with the themes of this thesis. The same is true for parts of the literature study in Chapter 3, as some of the subjects were relevant for both projects.

The part of the code written for the prototype called the "MQTT event handler" is a continuation of the code from the specialization project, which was based on the "simple_MQTT" sample from Nordic Semiconductor's nRF Connect Source Development Kit.

The rest of the work documented in this thesis are the author's contributions.

1.6 Disposition

Chapter 2 contains a description of Autronica's AutroSafe 4 system. First an overview of the system topology is presented. This is then followed by a more detailed description of hardware and protocols that are relevant to this project, along with a description of the tools and routines used during service and commissioning. A summary of the AutroLink v1 prototype from the specialization project is also a part of this chapter.

Chapter 3 is a literature study that covers other subjects that are relevant to the design and implementation of the new prototype. Subjects such as remote access and real-time systems are presented here.

Chapter 4 contains the analyses that led up to the new prototype's requirements. Furthermore, a structured analysis is conducted that separates the prototype's functionality into modules and submodules, followed by a summary of the design specifications.

Chapter 5 documents the design and implementation of the prototype. It also mentions how the development environment was set up.

Chapter 6 contains descriptions of the tests that were conducted on the finished prototype.

Chapter 7 consists of discussions regarding the results from the testing and other aspects of this thesis.

Chapter 8 concludes the thesis and presents items for future work.

Chapter 2

System description

This chapter contains a description of the AutroSafe system and it's components, Autronica's service and commissioning tools and routines, as well as a description of the AutroLink v1 prototype. The description of the AutroSafe system and the sections related to service and commissioning is based on both publicly available documentation and internal documentation from Autronica. The information here has also been expanded upon through dialogues with the engineers from Autronica's Research and Development (R&D) and service departments. The description of the AutroLink v1 prototype is summarized from the specialization project report. Some of the sections of this chapter were also covered in the specialization report, they have however been altered and expanded upon to better fit the theme of this thesis. New information has also been included where needed.

2.1 AutroSafe interactive fire detection system

The AutroSafe fire detection system, from here on referred to as AutroSafe, is a high-end system designed for large and complex buildings and to meet the requirements for onshore, maritime and offshore installations. In order to meet the strict requirements necessary on offshore facilities, the AutroSafe system has a SIL-2 certification [2]. SIL stands for safety Integrity Level, and is a standard for risk reduction. SIL-4 being the most reliable, while SIL-1 being the least. In order to have a system SIL approved, there are guidelines that need to be followed during the development and the safety life cycle management of the system. One of the most important requirements for continuously running systems such as a fire detection system is an attribute called Probability of Failure on Demand (PFD). In order to meet SIL-2, the system needs to have a PFD in the range $10^{-2} - 10^{-3}$.

A typical AutroSafe system consists of panels and controllers connected together via Autronica's Ethernet-based local network called AutroNet. One such unit that is mandatory in all AutroSafe installations is a Fire Alarm Control Panel, displayed in Figure 2.2. In some AutroSafe systems, a computer running AutroMaster ISEMS software is connected to the local AutroNet network. This is a command and control equipment that gathers information from all the panels connected to the AutroNet network and displays it graphically.

The fire alarm control panels and controllers can be connected to a module stack. The module stack is a set of equipment that expands upon the capabilities of the system. It can consist of devices that control detection loops (loop drivers), that control outputs (output modules) and detect inputs (input modules). The module stack itself cannot be seen in Figure 2.1, as it is usually mounted inside the cabinet

of a controller or panel.

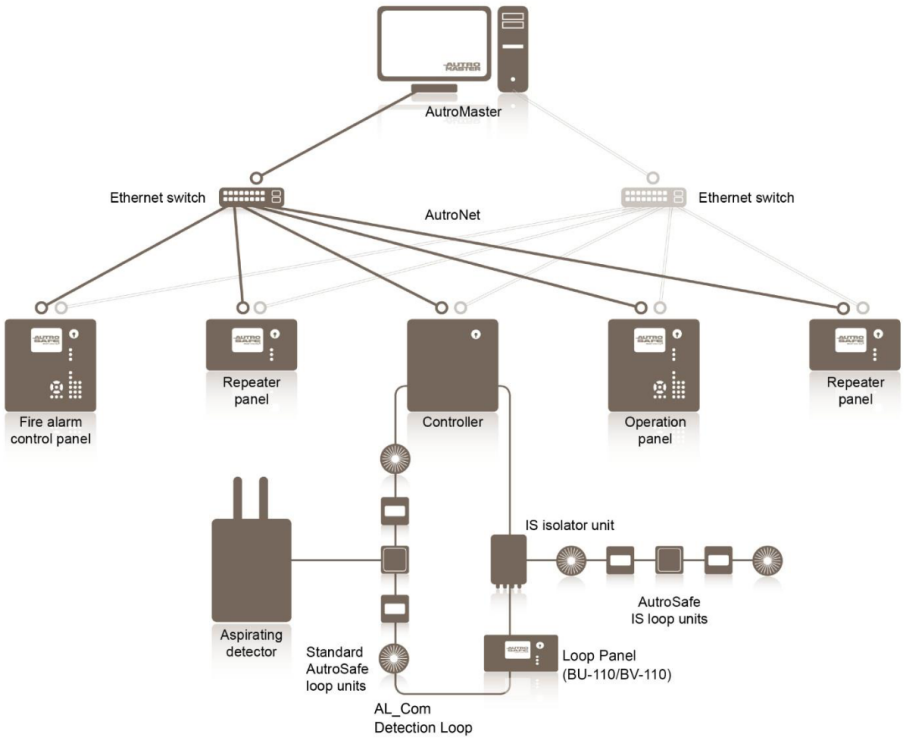

Figure 2.1: Sample installation of AutroSafe system, illustration from Autronica Fire & Security [2]

The detection loops consists of loop units such as smoke detectors, heat detectors and manual call points. The devices connected to the detection loop are commonly referred to as loop units. The detector loops can be connected to module stacks using the loop driver module [3]. A detection loop can be seen in Figure 2.1 connected to the central controller. This controller has a module stack containing a loop driver inside it's cabinet.

Different overall topologies are possible to design to fit with the type of installation, as seen in the AutroSafe 4 system description [2, p. 15-17].

2.2 Panels and AutroNet

The panels in the AutroSafe system provide a user interface. They also function as an access point into the settings of the system. The panels come in a variety of designs, with fire alarm control panels, operator panels, repeater panels and information panels. An image showing a fire alarm control panel can be seen in Figure 2.2. These panels are usually rack-mounted, or mounted on a cabinet which can house additional components, such as module stack units. Controllers and controllers can utilize the AL_Com+ port on their motherboard, and have a direct connection with a module stack, and subsequently a connection with loop drivers and loop units.

Figure 2.2: Fire alarm control panel BS-420, illustration from Autronica Fire & Security [2]

2.2.1 AutroNet

AutroNet is the communication protocol the panels in the AutroSafe system uses to communicate with each other. It is implemented over a redundant Ethernet connection, where switches connect the controllers together using Ethernet cables. Autronica strongly advises against connecting the local AutroNet network to the Internet, as this poses a potential cybersecurity threat [2, p. 14].

2.3 The module stack

The module stack can be customized to meet the specific installation's needs. The modules connect together physically by stacking on top of each other, connecting a set of female pins to male pins. This connects them together in a bus configuration. The modules also have a Light Emitting Diode (LED) and a photoransistor that are used for communication. The LED sits on top of each module, while the phototransistor is located on the bottom, lined up with the LED. This means that when a module stack unit activates its LED, the module located above it senses it as this activates its phototransistor. This is used for an addressing procedure described in Section 2.8.3. Two of the modules in the module stack are mandatory for the stack to function properly. The first one is the power supply module. It is connected to a 24V power supply and supplies the rest of the modules in the stack with 24VDC and 5VDC. The second mandatory module is the communication module, which enables the controller to communicate with the units in the module stack. It provides an access point in to the AL_Com+ bus through a 2x5 latch contact. Beyond this the module stack can be outfitted with loop driver modules, input modules and output modules. According to the AutroSafe 4 system description, a single module stack can have a maximum of 12 units as described in the system description of the AutroSafe 4 system [2].

2.4 The detection loop and loop units

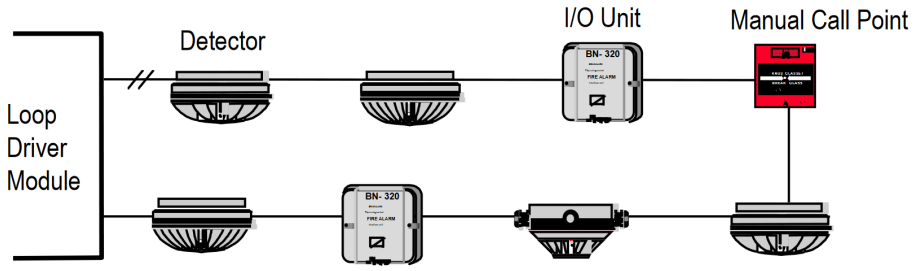

Figure 2.3: illustration of a detector loop, illustration from Autronica Fire & Security [2]

The detection loop is a series of detectors and other loop units connected to the loop driver module of the module stack via two wires. The two wires, "plus wire" and "minus wire", serve two purposes. They supply the loop units with +24VDC and are also used for the communication between loop units and the loop driver. An illustration of a detection loop connected to a loop driver can be seen in Figure 2.3. A maximum of 127 loop units can be connected to a single loop driver, as stated in the AutroSafe 4 system description [2]. Both ends of the two wires are

connected with the loop driver resulting in a loop of detectors. Along this loop, detectors can branch off from a node, as seen in Figure 2.4. This opens up for easier installation in some cases, additionally it can reduce cable and installation cost. However, the AutroSafe system only allows for a maximum of one branch-off on each detector on the loop. No more than 32 loop units can be connected in a single branch-off. This is in order to conform with EN-54 requirements of the consequences of a single-point failure [2].

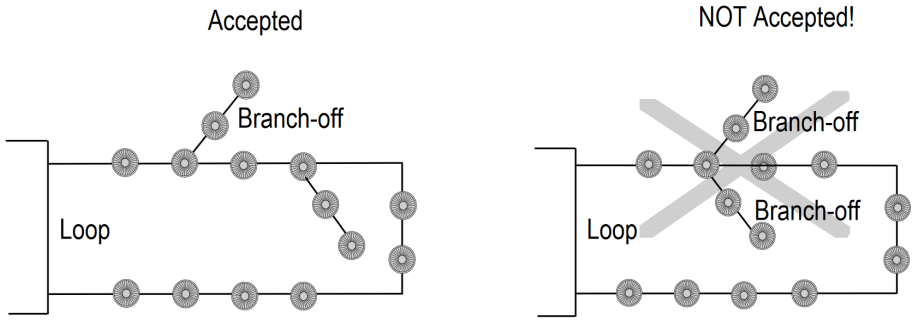

Figure 2.4: Illustration showing branches in the detector loop, illustration from Autronica Fire & Security [2]

Loop units are the devices that are connected to the detection loop. These consist of different kinds of detectors, manual call points, sounders and warning lights, to mention only a few. However, this chapter does not go into detail about the individual detectors as this is of little importance to the thesis. Autronica's detectors all have a socket which the detector is screwed into. In this socket, three screw terminals are connected to the detection loop. The plus wire has both the in-going and out-going wire connected to the same terminal, while the minus wire is connected to two different terminals. The detectors have a switch that by default is open, but can be used to close the circuit between the two terminals. The detectors use this switch when instructed by the loop driver. This feature is used during certain procedures, one of which is the loop raising procedure, which is described in greater detail in Section 2.8.4. While performing this procedure, each detector is also assigned a unique identificator called a "Loop ID", which let's the loop driver communicate with the individual detectors on the detection loop. This is necessary as the detection loop functions as a bus connection between all the detectors and the loop driver.

2.5 The AL_Com protocol

Autronica Loop Communication, or AL_Com for short, is both a hardware interface and a proprietary data protocol developed by Autronica. It is used in the communication between the loop driver and the units on the detection loop [4, p. 2].

When communicating, the loop units as well as the loop driver makes use of AL_Com. This protocol has a defined transmission rate of 1200 baud. When units on the detector loop want to communicate they short circuit the two wires. Logical 1 is represented by 0V on the detection loop, while logical 0 is represented by the quiescent voltage [5, p. 3]. This means that only one unit on the detector loop can transmit data at a time. This also goes for the loop driver, essentially making the AL_Com protocol simplex. In order to not lose power during the communication, each detector is outfitted with a capacitor that keeps the voltage in the detector high as the AL_Com loop is shorted.

The messages that gets sent on the detection loop are called directives. They are binary messages up to 16 bytes long. Each directive has a specific purpose. There are directives that the controller uses to request information from loop units, directives that are used to change settings and parameters in the detectors and directives that the detectors use to tell the controller of detected events. The last byte of an AL_Com directive is a checksum. This checksum allows the receiver of the directive to check for bit-errors. In order for units on the loop to receive the correct messages, all AL_Com messages contain a destination address and an origin address. All units will read the messages sent on the detection loop, since they are connected together via a bus. However, only the loop unit with the correct address will care about the message. There is also an option to broadcast messages. In this case, all loop units will care about the message. The loop driver has a fixed address, while the individual units on the detection loop can have their addresses changed. This is done through the use of a specific directive. There are two methods of verification to know whether a directive has been received by a loop unit. The first is when the loop unit sends an answer. One example of this is when the controller sends a directive to a loop unit asking it to identify itself. The loop unit will then respond with an identification directive. The other is by having the loop unit send an acknowledgment. Acknowledgments are requested by setting a specified bit in the directives to 1. Both the controllers and the loop units can request acknowledgment for their directives. Failure to respond to a directive with the acknowledge bit set high within 2 ms results in the directive being retransmitted [4, p. 3].

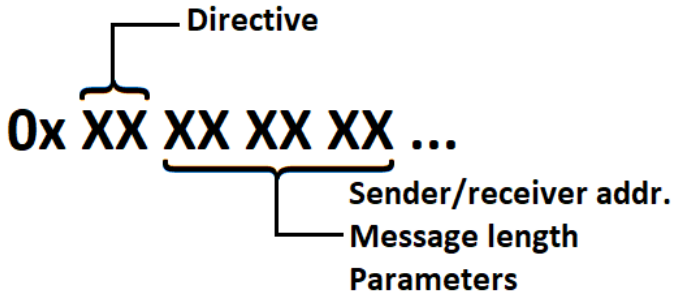

Figure 2.5: Structure of an AL_Com directive

If two loop units attempt to communicate at the same time collisions occur. The first loop unit to detect the collision will stop transmitting immediately. The collision gets detected when a unit attempts to send a logical 0, but reads that a logical 1 gets sent during this transmission period [5]. Retransmission takes place 100 ms after the previous message for most messages, but only 35 ms for messages with higher priority. This is done so that more important messages reach their destination faster. These are messages such as directives reporting on alarm conditions, directives to control sounders and I/O, as well as answers and acknowledgements to directives with priority. The loop driver has an even higher priority than the loop units. It will retransmit regular messages after 90 ms and high priority messages after 33 ms.

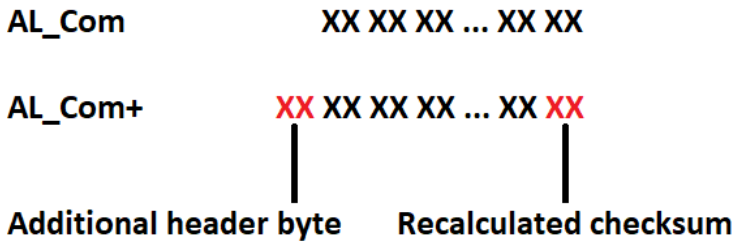

Figure 2.6: Difference between an AL_Com and an AL_Com+ directive

2.6 The AL_Com+ protocol

AL_Com+ is an extension of the AL_Com protocol. It is used in the communication between a panel and the modules of its module stack. This enables messages to pass back and forth between the panel and the loop units, through the loop drivers.

The messages that get sent over AL_Com+ can be separated into two main categories, directives and flow control messages. As shown in Figure 2.6, the AL_Com+ directives have an added header byte, which contains information about which module stack unit the directive is meant for. Additionally, the checksum that is part of the AL_Com directive gets re-calculated with the header byte in mind. This allows the controller to send messages to specific loop units through the loop drivers. There are also some added directives that only get passed between the controller and the units in the module stack. These directives are used for configuring of- and information gathering from the units in the module stack. This is illustrated in Figure 2.6, where each "X" represents a hexadecimal value, and "XX" makes up one byte of data.

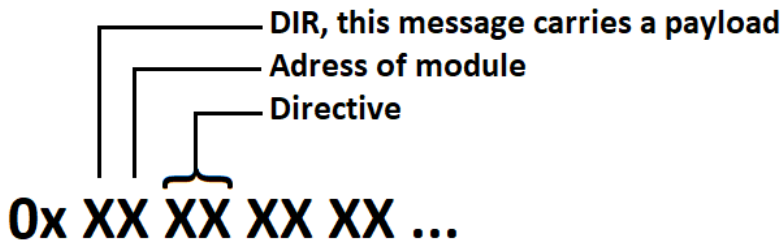

Figure 2.7: Structure of an AL_Com+ directive

The AL_Com+ protocol uses flow control messages to manage the transmission of messages between the controller and the units in the module stack. These are the remaining entries in the table below. As with the directives in Figure 2.7 and Figure 2.6, **XX** represents one byte of data.

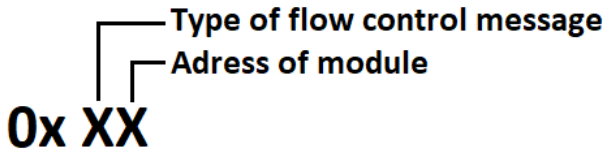

Figure 2.8: Structure of a flow control message

NOP Nothing to transmit

ENTX Poll/request information

ACK Acknowledged

NACK Not Acknowledged

DIR Directive, followed by a payload

BUSY Used to signal that the unit is currently preoccupied

The AL_Com+ protocol is implemented using the RS-232 standard for communication with a symbol rate of 9600 baud. The transmit and receive lines behave according to the standard. However, the control lines Request To Send (RTS) and Clear To Send (CTS) do not. Instead of using these two signal lines for flow control, they are used for specific events in the system. From the controller's point of view, the RTS signal is used to initiate the addressing sequence of the module stack. This is explained in greater detail in Section 2.8.3. The controller monitors the CTS line, as this is used by the modules in the module stack when they have an important directive they need to send that wasn't requested. Some examples of directives that utilize this feature are wake-up messages, restart messages and alarm directives.

2.7 System configuration

In the AutroSafe system, a configuration file determines the valid topology of the system. The configuration and the physical installation need to match fully for the system to operate without fault. If an event that would alter the system occurs, the system will report it as a warning. These events could be malfunctioning detectors, loop-wire getting cut or shorted, detectors taken from their sockets getting put back in the wrong socket, a branch-off being made for additional detectors. If changes like these are not represented in the configuration file, it would result in the controller raising it as a fault.

2.8 Commissioning and service

This section contains information about the existing tools and routines Autronica uses when performing commissioning and service on their AutroSafe systems. This information will later be analyzed in order to identify aspects of the tools and routines that a prototype such as the AutroLink v1 could improve.

The commissioning of the system usually happens at the very end of the installation phase. However, in larger installations these two phases may overlap. After the system has been installed, a set of procedures are performed on the system in order to get it up and running. When the system is running, and up until the system is decommissioned it will occasionally require service, expansion or changes. Service in this context means to locate and eliminate errors, replace broken or worn out equipment and perform checks on the system at regular intervals. This applies up until the system's end of life. The following sections describe the most common tools and routines used during service and commissioning.

2.8.1 AS-2000 Loop Diagnostic Tool

The AS-2000 Loop Diagnostic Tool is a computer program used to analyze the detection loops in an AutroSafe system. It is able to communicate with module stacks and detection loops by deploying the same procedures performed by the AutroSafe system [6]. Figure 2.9 shows a computer running AS-2000 connecting with a detection loop through the module stack present in the control panel cabinet.

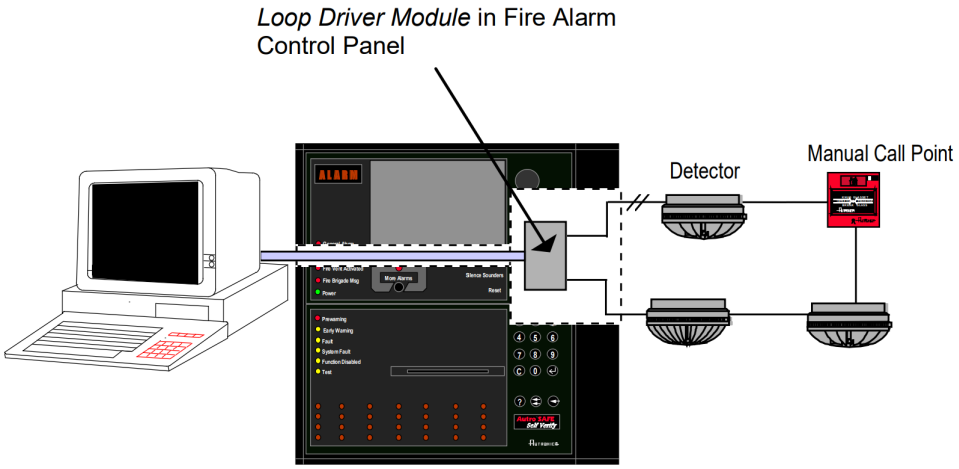

Figure 2.9: A computer running the AS-2000 Loop Diagnostic Tool software connecting with a detection loop through a panel, illustration from Autronica Fire & Security [6]

In this configuration, the panel is first disconnected from the communication module of its module stack. A cable, as seen in Figure 2.10, is then used to connect the one of the computer's COM ports to the 2x5 latch connector of the communication module.

Figure 2.10: Cable used to connect a computer with the AutroSafe system's communication module, image courtesy of Autronica Fire & Security

Before communicating with a detection loop, the AS-2000 software needs to know which modules are present in the module stack. To achieve this, AS-2000 performs a procedure that assigns addresses to the modules in the module stack. Through this procedure, AS-2000 gathers information about the individual modules and is able to tell which module is a loop driver. This also makes it possible to communicate with the individual modules. This procedure is explained in greater detail in Section 2.8.3.

Once the module stack has been addressed and a loop driver has been selected as the target module, it is possible to start the loop raising procedure. Through this procedure, AS-2000 gathers information about the individual loop units connected to the loop driver. As AS-2000 is performing the loop raising procedure, it will consecutively display the discovered detection units and the topology of the loop graphically. This procedure is explained in greater detail in Section 2.8.4. As stated in Section 2.4, more than one branch off is not permitted in an AutroSafe system. If this is the case with the detection loop being raised by AS-2000, an error will be displayed.

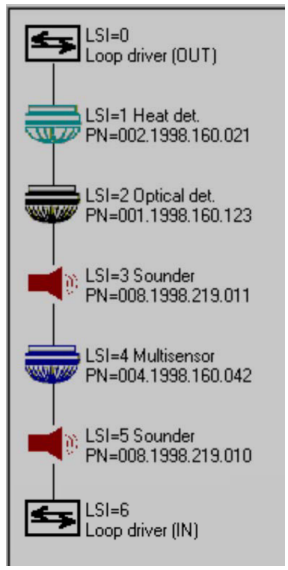

Figure 2.11: Screenshot from the AS-2000 Loop Diagnostic Tool showing the topology of a detection loop, image from Autronica Fire & Security [6], edited for clarity

With the loop raised it is possible to retrieve information about the detectors, such as loop unit type, production number, software version and condition. It also allows service engineers to use AS-2000 to start various procedures.

One of these procedures is an LED test procedure. This allows service engineers to verify the position and address of the loop units by lighting the LEDs and locating them throughout the facility. Another procedure is one that is able to detect breaks and short circuits along the detection loop. Additionally, measurements of the detection loop, such as loop resistance, current consumption and voltage drop can be performed through AS-2000.

2.8.2 WAS-2000 Service-suitcase

The WAS-2000 service-suitcase serves as a tool for service engineers used during service and commissioning of the AutroSafe system. It contains a module stack consisting of three modules. A power supply module, a communication module and a loop driver. There is also a 24V battery connected with the module stack. The WAS-2000 is used to be able to disconnect a single detection loop from an AutroSafe system in order to perform service on it, without affecting the rest of the AutroSafe system. This way, only one detection loop gets disconnected from the system. The rest of the units on the AutroSafe's module stack can function as normal.

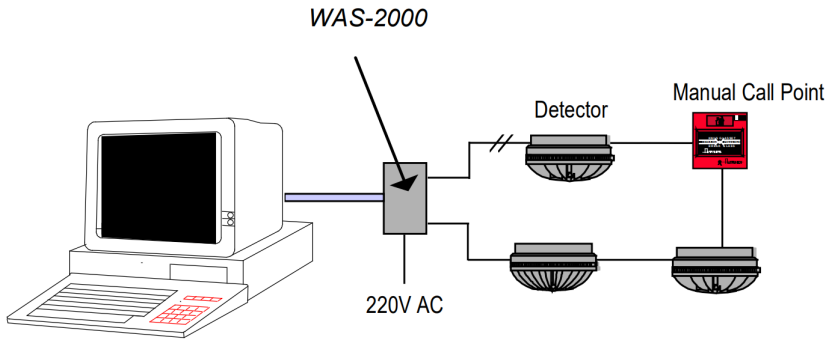

Figure 2.12: A computer running the AS-2000 Loop Diagnostic Tool software connecting with a detection loop through WAS-2000, illustration from Autronica Fire & Security [6]

2.8.3 Addressing the module stack

Module stack units are connected together via a bus. In order for the controller to be able to communicate with the individual modules, the modules require unique addresses. They receive these addresses through a specific procedure. This procedure utilizes the RTS signal in AL_Com+, the LED on the top of the modules, and the phototransistor on the bottom of the modules. While the RTS signal is activated and before a module stack unit has received an address, it does not respond to **ENTX** messages unless the phototransistor is activated by the LED of the previous module. This is true for all modules except the first addressable module in the stack, the communication module. Therefore, when activating RTS and sending an **ENTX** message the communication module will be the only one to respond. When this module receives its address it activates its LED, meaning the next time the controller sends an **ENTX** message, the module above the communication module will be the one to receive an address and respond. However, during this procedure the controller needs to assign unique addresses to each module. It does this by keeping track of the number of modules that respond and changing the address nibble of the flow control message each time. The address nibble of the **ENTX** message can be seen as the final **X** in Figure 2.8. The following list shows the addressing procedure as enumerated steps.

1. The **RTS** signal is activated
2. An **ENTX** + `device_number` message is sent from the controller to the module stack
3. The i'th module takes `device_number` as its address
4. The module stack unit responds with **NOP** or **DIR** + `device_number`

5. If the response was a **DIR**, this gets acknowledged by the controller with an **ACK** + `device_number`
6. `device_number` is incremented
7. Steps 2 through 6 is repeated until no response is received after sending an **ENTX**
8. The **RTS** signal is deactivated

The same procedure is shown graphically in Figure 2.13. The controller to the left in the figure can be a panel in the AutroSafe system or the AS-2000 Loop Diagnostic Tool. The stream of messages are chronologically ordered from top to bottom. An arrow pointing from the controller to the module stack represents a message sent from controller to module stack, and vice versa. The dotted parts of the vertical arrows represents that the previous steps can be repeated as many times as there are remaining module stack units. The second to last event in the figure is a timeout event triggered by an internal timer in the controller.

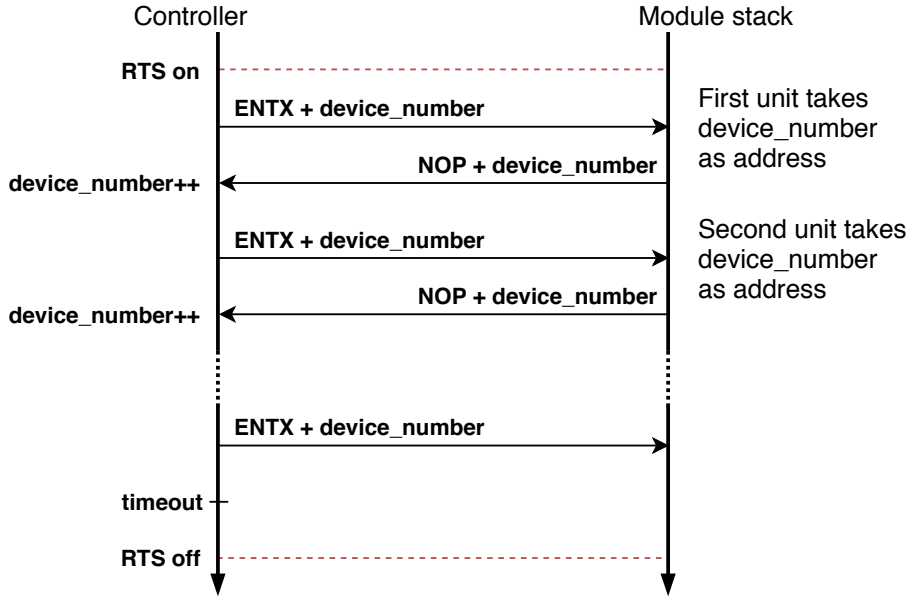

Figure 2.13: Illustration of the communication between a controller and the module stack during the module stack addressing procedure

2.8.4 Raising the detection loop

Initially, after powering up an AutroSafe system and addressing the module stack, the detection loops are voltage free. This means that none of the detectors are active, and the switch mentioned in Section 2.4is open, as seen in Figure 2.14. A

procedure called “raising the detection loop” powers the detection loop and assigns unique loop ID’s to the detectors.

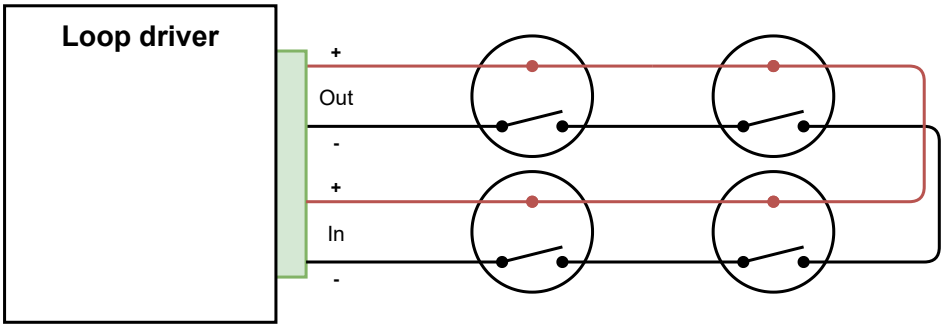

Figure 2.14: Illustration showing detectors connected to a loop driver

The procedure starts with the controller sending a directive to the loop driver instructing it to apply power to the detector loop. This will in turn power the first loop unit, making it transmit a start-up message. This message contains information such as current loop ID, serial number and point type. The controller can then send a message to the first loop unit instructing it to close its switch, which will power the next loop unit on the detector loop. These steps are repeated until the loop driver receives power in it’s second set of terminals. During this procedure, the controller needs to keep track of which Loop ID’s are in use. If two detectors have the same Loop ID the controller needs to assign a new Loop ID to one of them.

1. Controller sends rise loop **DIR** to loop driver instructing it to apply voltage to its OUT terminals
2. Next detector on loop receives power, sends restart **DIR**
3. Controller receives restart **DIR**, sends **ACK**, assigns new loop ID if necessary
4. Controller sends point switch **DIR** to the detector instructing it to close the switch
5. Detector closes switch
6. Steps 2 though 5 are repeated until loop driver receives power on its IN terminals
7. Loop driver sends loop closed **DIR** to controller
8. Controller sends loop output states **DIR** (normal loop) to loop driver

In the event of a branch-off, two detectors will send restart directives as part of step 2. The controller then needs to store both messages and perform the following steps on both branches until one of them results in a closed loop and the other results

in no new restart message after step 5, signifying the end of the branch. At this point the loop is considered "raised". It stays this way until the controller issues a directive to power the loop down, the loop driver loses power, or an external event breaks the loop.

2.8.5 Commissioning routine

The commissioning routines for the AutroSafe system is documented in the AutroSafe Commissioning Handbook [7]. The standard commissioning routine is separated into ten steps. [7, p.7-37]

Step 1

The first step during the commissioning of an AutroSafe installation is to verify the detection loops. This is done through the use of the AS-2000 software and performing the loop raising procedure explained in Section 2.8.4.

Step 2

In this step, the information gathered during the loop raise in step 1 is compared to the system's configuration file. The AutroSafe Configuration Tool software is used to perform this task.

Steps 3 through 5

These steps all consist of assigning addresses and ID's to the different components of the system. AutroFieldBus units and power boards receive unique addresses by the use of dip-switches. The panels and controllers throughout the system receive both a network ID and a panel ID by the use of rotary switches located on the back of the motherboard of the panels.

Step 6

This step is the second verification step. Here, all components and cables are checked to verify that the system has been installed correctly.

Step 7

In step 7, power is applied to the system. The system then performs an automatic initialization. During this initialization, the system detects whether the AutroNet network has a star- or ring topology. Alternatively, if the topology is neither star or ring, an error is raised which must be solved before continuing.

Step 8

After the system has been powered up, service engineers access the panel settings. Here the panel's IP address for the local AutroNet network is set. This is either done through an automatic process or it can be done manually.

Step 9

In this step, the finished configuration file is uploaded to the system. This can be done from any of the panels in the system, as it will distribute the file to the other panels connected to the same local AutroNet network. It can also be uploaded through a web interface. A memory stick is connected to a USB port located on the back of a panel. The service engineer can then log in to the panel with access rights and upload the configuration file to the system from the memory stick.

Step 10

The last step is to perform the third and and most thorough verification of the system. The system's alarm transference functions are temporarily disabled, as to not trigger emergency response from the fire-brigade during the testing. The panel's indicator lights, buzzer, operating keys and functions are tested. System alarms and sounders are tested. Detecting loop units in each detection zone are tested. Fault indication is tested by generating faults in the system, such as removing a detector from it's socket. The system's different conditions are tested (test, disablement, fault warning, fire warning and fire alarm conditions). Once the test are completed the system is set to normal operation, and the alarm transference is enabled.

2.9 AutroLink v1

During the specialization project, the AutroLink v1 prototype was designed, implemented and tested. The goal of this project was to gather data from the AutroSafe system during normal operation and send this wirelessly to an Internet-connected server [1].

As the prototype was to be based on the nRF9160 development kit from Nordic Semiconductor, it natively supported the Long-Term Evolution Machine Type Communication (LTE-M) and Narrowband Internet of Things (NB-IoT) cellular networks. By comparing the two networks it was found that the LTE-M was more fitting for the prototype due to the availability, bandwidth and roaming capabilities.

The possible points of data extraction in the AutroSafe system were analyzed. In the end, the prototype connected with a listener port located on the motherboard of the panels in the AutroSafe system. The port provided access to the AL_Com+ communication passed between the panel and its module stack. The listener port had two pins, one being ground and the other being both the transmit and receive signal of the RS-232 line between the panel and the module stack. These were connected through diodes, meaning that by connecting to the port the prototype would receive both the messages sent to and from the module stack. It also meant that the prototype was physically limited to only receive messages from the port, unable to inject AL_Com+ messages and other data into the system.

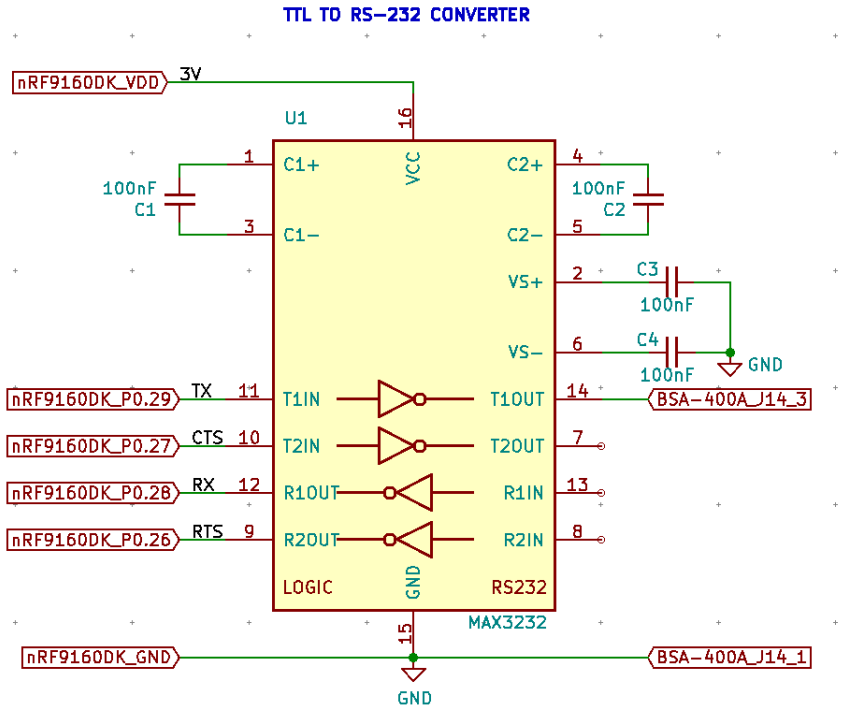

Figure 2.15: TTL to RS-232 conversion using a MAX3232 integrated circuit, schematic made using KiCAD EDA [1]

The finished AutroLink v1 consisted of the nRF9160 DK connected to an external PCB containing a MAX3232 circuit. This circuit converted back and forth between RS-232 and Transistor-Transistor Logic (TTL) voltage levels and communication. The connections were made using jumper wires from the PCA-10090 board's General-Purpose Input/Output (GPIO) over to the MAX3232 board. The MAX3232 was powered by two jumper wires from the nRF9160 DK GPIO's 3V and GND GPIO pins, seen in Figure 2.16 as the red and black wires respectively. The Transmit (TX) and Receive (RX) signals were carried by the blue and purple wires, although only the RX signal lines was tested and used. These were the minimum hardware components for the prototype to function and interface with the AutroSafe system. The prototype connects with the AutroSafe system's module stack by connecting an RS-232 DB9 to 2x5 latch connector cable, as seen in Figure 2.10, to the COM port located on the uppermost PCB marked MAX3232 in Figure 2.16. It was supplied with 5VDC from an external power supply.

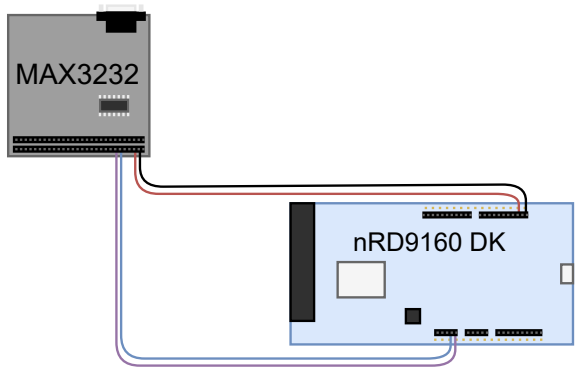

Figure 2.16: Illustration of the PCA-10090 board connected to the MAX3232 circuit board

The finished prototype was tested by connecting it to an existing AutroSafe test station at Autronica's R&D department. A Mosquitto Message Queuing Telemetry Transport (MQTT) client was set up on a computer [8]. Both the AutroLink v1 and this client connected with a MQTT server, as seen in Figure 2.17. The AL_Com+ messages transmitted from and to the module stack were picked up and sent to a specified topic. Using another topic, a Mosquitto client was used to send configuration changes to the AutroLink v1 prototype.

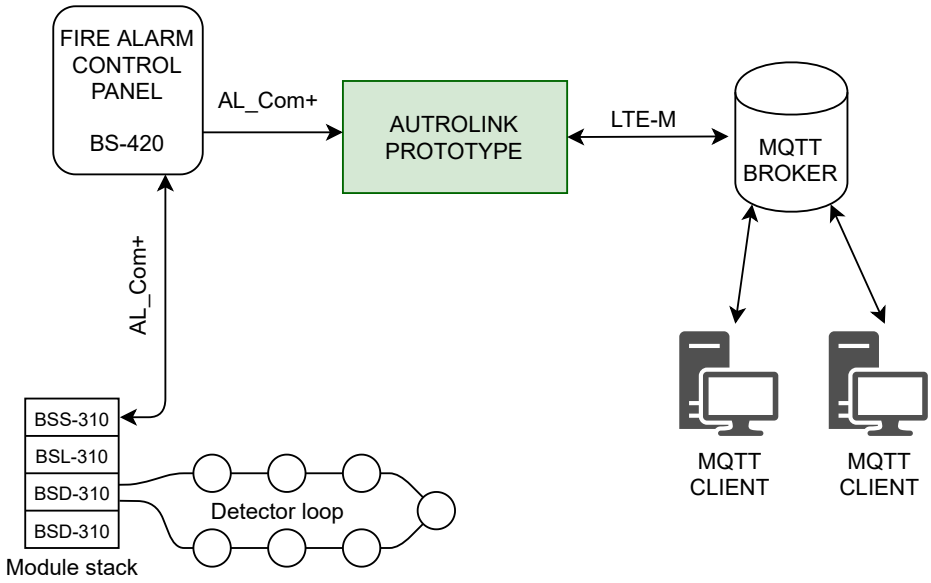

Figure 2.17: An overview of the AutroLink v1 prototype connected with the AutroSafe and the MQTT server

Chapter 3

Literature study

This chapter summarizes the topics that were studied that are not related to the AutroSafe system. The first section contains information about remote access solutions with a focus on solutions regarding remote service and commissioning. The following sections contain information on real-time system design, the MQTT protocol and Nordic Semiconductor's software development kit.

3.1 Remote access

Remote access is a term closely related to computer systems, wherein an authorized user is able to access a centralized system through a network. Traditionally, this has been used to control or configure computers or servers, but in the wake of Industry 4.0 the range of applications has been broadened. One such application is what is being explored in this master's thesis, namely adding remote service and commissioning solutions to existing systems. Through the literature study on service and commissioning with remote access, few sources that were of relevant technical detail to this project were found.

In 2017, Riccardo Masoni et al. wrote a research paper exploring the use of Augmented Reality (AR) in remote maintenance. The paper describes a previous prototype that was built and documents the work that went into improving it based on feedback from the initial testing. The prototype consisted of off-the-shelf mobile and AR technologies, making it accessible and cost effective [9]. An advanced tool like this can convey information in a way that makes it more accessible, and bring in expert knowledge to assist during the service and commissioning routines. In essence, remote access allows an expert to access necessary parts of the system without having to travel to the actual location of the facility. This can expedite the service routines, which in turn can be more cost effective than alternative solutions. The technical details present in this research paper are of little relevance to this thesis.

The search terms that were used produced several hits on advertisements for proprietary remote access and remote maintenance solutions. An interesting detail was that several of them were announced recently due to the Covid-19 pandemic. These sources consisted of product descriptions from manufacturers, descriptions of services from various providers and blog posts. Schenck Process introduced remote commissioning of their industrial machines [10]. Coperion modernized their service routines by developing their "Coperion ServiceBox" which connects with their industrial plastic extruders [11]. The aforementioned examples were however primarily proprietary solutions that are not described in technical detail. They are

not considered as viable sources, but they do prove that proprietary remote access solutions are prevalent in the industry.

One industry that has adapted the use of remote access technology is the oil and gas industry. This was done with the aim of making the monitoring, operation and service of the on-board systems cheaper and more efficient. With oil rigs and platforms located at sea, transport costs are at a premium. However, by introducing remote access to a system, the system also becomes exposed to possible cybersecurity attacks. Therefore, it is necessary to integrate remote access solutions in a way that is secure. Sintef and DNVGL have written reports on this specific subject. However, these sources are abstracted away from technical solutions, and instead describe concepts and best practices.

3.1.1 SeSa method

The report from Sintef documents a method for assessing remote access in Safety Instrumented System (SIS), called the SecureSafety (SeSa) method. It describes a systematic approach to assess whether a given technological solution for remote access to the SIS implies an unacceptable risk, in terms of jeopardizing the Safety Integrity Level (SIL) of the SIS. The report is based on the security standards IEC 61508, IEC 61511, ISO 15408 and ISO/IEC 27001.

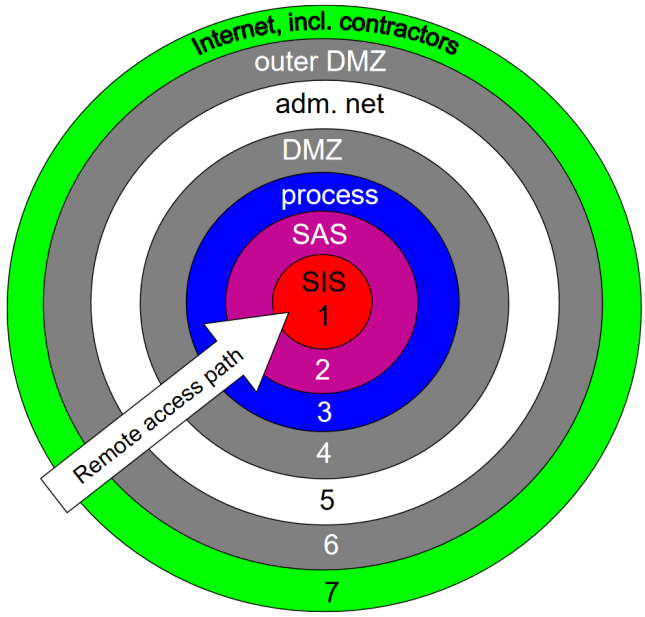

Figure 3.1: Onion model depicting a layered network architecture, illustration from Sintef [12]

The workflow when following the SeSa method is described in the flow chart in Figure 3.2. If a suggested remote access solution does not have an impact on the SIL of the related system it is deemed "ready" for implementation. If it does impact the SIL, new functions are suggested where necessary, and new requirements are specified for the security functions. If the revised solution still results in an impact in the SIL, security functions in the SIS are analyzed. A Hazard and Operability study (HAZOP) is then performed to check whether the SIS's security functions are sufficient. If not, the proposed remote access solution is discarded all together.

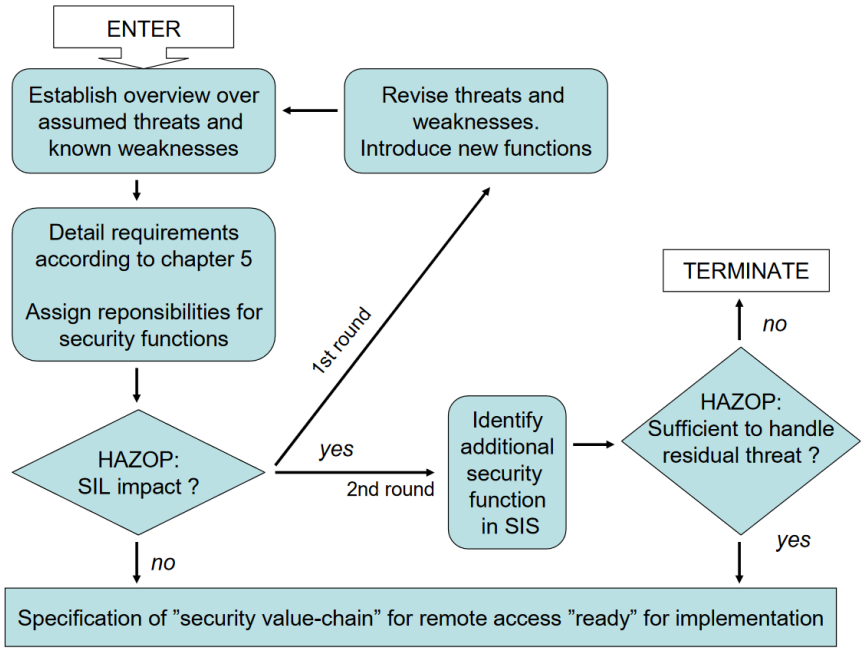

Figure 3.2: Flow chart of the SeSa method, illustration from Sintef [12]

3.1.2 DNVGL RP-108

The report from DNVGL is thematically similar, but is aimed at Industrial Automation and Control System (IACS). It contains guidelines for how to apply the IEC 62443 standard with the strict requirements in the oil and gas industry. After a read-through, the most relevant sections were selected and are summarized below.

The report has a section pertaining temporary connected devices. These are devices such as engineering tools, configuration tools and diagnostic tools. Some of these may be necessary to operate the systems while others are just used during maintenance of the system. The report also separates the considerations that should

be made regarding this equipment into two categories based on the criticality of the particular system. For highly critical systems, the report recommends including these tools as permanent equipment. However, if the criticality of the systems are low, the tools may be connected as temporary devices. Procedures should be created for when it is external suppliers that need to use temporary devices to verify the safety of the equipment. One should ensure that the equipment is using the latest patches, is sufficiently hardened and has the latest antivirus if possible. External suppliers should always need permission from the site owner before they connect temporary devices, and the connection should be handled as a deviation from normal operation [13, p. 21].

The report goes into detail about how solutions for remote access should be implemented. Due to large distances, costly transport and safety requirements, it is practical to allow remote online access for maintenance. By doing this, one bypasses physical security and authentication and a large cybersecurity attack surface is exposed. It is therefore vital to design and build a secure solution for this remote access. The remote access solution should also have some form of identification and authentication control. Multi-factor authentication is recommended, as well as a timeout for inactivity. In order to protect the information that gets sent along the remote access path, encryption should be used. A recommendation for symmetric encryption is AES 128 or better [13, p. 32].

3.2 MQTT communication protocol

MQTT is a messaging protocol for use on top of Transmission Control Protocol (TCP)/Internet Protocol (IP). This is the communication protocol that was implemented in the AutroLink v1 prototype in order for it to communicate with an online server, and through it reach other MQTT clients. It is widely used in Machine to Machine (M2M) and Internet of Things (IoT) communication since it is lightweight and easy to implement. It uses a publish-subscribe messaging pattern where clients connected to a central broker can publish and subscribe to topics [14]. Whenever a message gets published to a specific topic, the broker passes this message to every client who is subscribed to said topic. An illustration of this principle can be seen in Figure 3.3, where three clients are subscribed to the same topic. A fourth client publishes a message to said topic and the broker handles the distribution of the message.

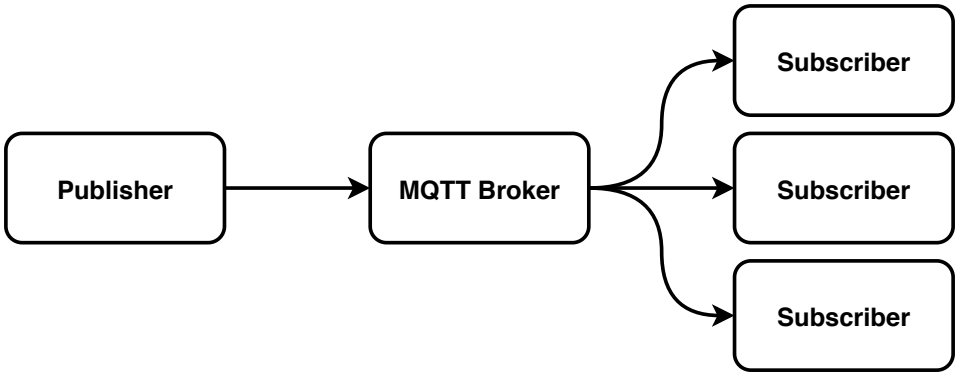

Figure 3.3: MQTT messaging pattern

The protocol have been implemented by different groups, such as Zephyr [15] and Mosquitto [8], who have libraries that support it.

3.2.1 Topics

MQTT has a channel based structure, with the communication divided into self-defined topics. A topic is made up of one or more topic levels. These levels are separated by a forward slash (`/`), a topic level separator. Each level can be named to describe what kind of information gets published on it. Below are three sample topics that are used on the the same MQTT broker.

1. `Location_A/System_A/Sensor_A`
2. `Location_A/System_B/Sensor_A`
3. `Location_A/System_B/Sensor_C`

If an MQTT client is subscribed to the first topic, it will receive all published messages on this topic. The MQTT topic structure supports wildcards. The single-level wildcard (`+`) can be used to substitute one level. As an example; a client subscribing to: `Location_A/+/Sensor_A` would receive the messages published on both the first and the second topic. The multi-level wildcard (`#`) can be used as a substitute for more than one level. It can only be used at the end of the topic string. A client subscribed to: `Location_A/#` would receive the messages published to all the aforementioned topics. A single client can publish and subscribe to different topics. This makes the messaging pattern of MQTT highly versatile.

3.2.2 QoS

The MQTT protocol provides three different Quality of Service (QoS) levels [14].

QoS 0 "At most once", the message gets published once with no guarantee that the message arrives. This may result in lost messages, and is most commonly used in sensor readings where one lost message has little to no impact on the functionality of the system.

QoS 1 "At least once", the message gets published repeatedly until the publisher receives an acknowledgment from the subscriber that the message has arrived. This may result in duplicated messages.

QoS 2 "Exactly once", the message arrives exactly once through a series of steps. This is however also the slowest form of MQTT communication due to the amount of handshakes involved.

3.2.3 TLS

Transport Layer Security (TLS) is a cryptographic protocol designed to provide security in the application layer of the Internet protocol suite. It supports exchanging keys, data encryption, and message authentication [16]. As such, this protocol can help with data confidentiality, and help prevent certain attacks, such as man-in-the-middle attacks [17]. It is possible to implement TLS in MQTT. Nordic Semiconductor have, among others, support for it in their MQTT applications [18].

3.2.4 Mutual authentication

Another layer of protection that is common in other communication protocols is mutual authentication. In mutual authentication, both the server and the client exchange certificates. This lets the server verify the identity of the client, and the client the identity of the server. Although not as usual as TLS, mutual authentication has also been implemented in MQTT communication. A publically available example of this is freeRTOS' demo [19].

3.3 Real-time systems

A real-time system has fixed constraints where processes must be done within their limitations or the system will fail. This means that the response of the system needs to meet a specific time restraint, or need to meet some form of deadline. In other words, the system not only needs to produce the right solution, it also needs to provide it at the correct time. The AutroLink v2 prototype needs to interface with the AutroSafe system, giving it constraints in form of time-dependencies. It can therefore be viewed as a real-time system itself, and there are certain aspects of real-time programming that is necessary to keep in mind then designing and

implementing such a system. Real-time systems are commonly divided into two categories, by the consequences of missing the deadline [20].

Hard real-time system In hard real-time systems, the consequences of missing deadlines can range from bad to disastrous. It is characterized by that the value of the output from the system drops from needed to worthless. An example of this is the brake-system of a car, where it is of little use that the brakes activate if the delay already have caused a collision [21].

Soft real-time system Soft real-time systems are characterized by the decreasing value of the system output. This means that the output is of less relevance after the deadline, but still have some use depending on the amount of delay. An example of this could be a GPS system, where a delayed output does not give your current location, but can be used to determine where you have been, and if your general direction of movement is correct [21].

Types of tasks

In real-time system's theory, tasks are usually organized into three categories:

Periodic: Tasks that are repeated at a regular interval.

Aperiodic: Tasks that have no defined interval, and can thus appear at any time.

Sporadic: Tasks that also have no defined interval, but there is a minimum time between each occurrence.

3.3.1 Priority

If a system only has one output to produce, it can dedicate all of its resources to that task. However, real-time systems often has to perform multiple tasks where each has a deadline. To complicate things further, each task may have a different grade of consequence if the deadline is missed. An example of this can be that it is more important to ensure that the breaks of the car activate in time than that the breaking lights are turned on.

A way of solving this problem, is to assign a priority to the tasks a system has to perform. This can be a static priority where some tasks are designated as more important than other tasks, and therefore has to be done first. It is also possible to implement dynamic priority, where priorities can shift after some property of the tasks, such as what task has the shortest time to deadline. However, implementing priority can also lead to additional complexity and problems. A problem with prioritized tasks is that starvation of processes can occur. This is when the amount of higher priority tasks causes the system to ignore the lower priority tasks, as there is simply not enough time or resources to handle all the tasks.

3.3.2 Multi-threaded systems

As the amount of tasks to be done increases, a traditional sequential program may grow exponentially complex in order to handle the timing requirements of the different tasks. A way to solve this is by defining each task as a thread, and use an operating system to manage the execution of these threads. A simple version of this is a round robin implementation that does a bit of work on a thread before shifting to another thread in a loop. This allows tasks to be done concurrently and independent of each other, which simplifies the program structure.

However, when the complexity of the interwoven tasks are removed, another problem takes its place. When tasks are done concurrently, they may interfere with each other in a way that causes unwanted behavior. An example of this can be that one thread updates a value that is in use by another thread that is in the middle of an calculation. This may result in a wrong output. In addition, these kind of errors may occur seemingly at random, as they are dependent on when tasks are done. This particular problem are an example of a race condition, where the outcome is affected by the sequence the tasks are done in.

3.3.3 Inter-thread synchronization

In systems where there are multiple threads running, some processes may need to finish before others. There may also be other cases where two threads need to use the same resource in a system. Inter-thread synchronization is used in these cases. The most commonly used tools to achieve this are the variable types called semaphores and mutexes. Conceptualized by Edsger Dijkstra in the early '60s, the semaphore is an abstract data type used by threads to signal the scheduler. It can have any positive integer value. When a thread decrements a semaphore its value is decreased by one. If the value is zero as a thread attempts to decrement it, the thread blocks itself and cannot resume execution until another thread increments the semaphore. The mutex is essentially a special type of semaphore where its value can only be 1 or 0. It is normally used to ensure mutual exclusion, which is where the name comes from [22].

It is however possible to misuse semaphores. A semaphore may be requested, but not released. It may be released without ever having been requested. Additionally, situations such as deadlocks and livelocks can occur.

Deadlocks

Deadlocks are the therm for when two or more tasks are dependent on a progress of the other task in order to be able to progress itself. A common example of this is two tasks that both require two resources, A and B, before they can be executed. If one task has locked the resource A, while the other task has locked resource B, they will both wait forever on each other as no resource will become available before one of the tasks are executed.

Livelocks

A livelock differs from a deadlock in that the threads can change states or do some work, but no progress is done. A common example of this may be two persons meeting in a hallway, where both move to the side to allow the other to pass. In this way, they continually block each other but are still active.

3.3.4 Time complexity

Time complexity is a term in computer science that describes the computational complexity of an algorithm. This is usually done through analyzing the amount of elementary operations the algorithm has to perform for a given input. This analysis is abstracted from time, since different hardware can perform the same elementary operations at different speeds. This is affected by properties such as the type and the frequency of the processor. Since real-time systems need to guarantee response within specified time constraints, knowing the time complexity of the system's tasks is important.

Big O notation is commonly used to express the time complexity of an algorithm, given an input n [23]. Here, n represents the size of the input, e.g. the length of an array or the size of a matrix. The algorithm will then perform a set of tasks on the input. The types of tasks performed will have an impact on the execution time of the algorithm. The resulting growth of the execution time can then be expressed using the following expressions:

$o(g(n))$ denotes fewer than (<) $f(n)$ iterations

$O(g(n))$ denotes fewer than or equal to (<=) $f(n)$ iterations

$\Theta(g(n))$ denotes equal to (=) $f(n)$ iterations

$\Omega(g(n))$ denotes greater than or equal to (>=) $f(n)$ iterations

$\omega(g(n))$ denotes greater than (>) $f(n)$ iterations

In computer science $O(g(n))$ is most commonly used. The reason for this is that it provides an upper bound. In other words, it expresses the worst case execution time. $g(n)$ expresses the tail behavior of the algorithm, and is usually simplified to only consist of the largest factor of the expression.

When evaluating the time complexity of an algorithm, one usually looks at the number of elemental operations and loop structures. An elemental operation is a task that has a constant execution time, such as evaluating if two number are equal. For an instance, imagine a function takes an array as input and iterates through the elements using a loop. These operations lead to the function having $O(n)$ execution time, since the operations have to be performed on each element of the input array. The longer the input array is, the longer the function will take to complete. This demonstrates a linear growth rate. If the function would iterate through the array once per element of the array, it would have an execution time of $O(n^2)$, which is an example of an exponential growth rate. If a function takes

advantage of a "divide and conquer" approach, it usually has an execution time of $O(n \log n)$, which is called a logarithmic growth rate [23].

3.3.5 Crash-only software

When designing real-time systems, there are many different possible design methodologies to follow. Fault-tolerant systems are designed so that if it experiences a fault, there are certain safeguards in place that allow it to keep running. Fail-safe systems are designed in a way where most conceivable problems are accounted for. On the other end of the spectrum exists crash-only systems. These systems are designed to crash as soon as it enters an unknown state or encounters an unexpected result. At first glance this may seem counter-intuitive. However, in a system that itself is not safety critical, this is a form of fault handling. As long as the system is able to restart itself and enter a known state, normal operation can be resumed. By designing a system according to this mentality, even problems that are unforeseen can be handled [24].

3.4 Software Testing

Software testing is a method of checking if the software product is behaving as expected or according to specifications. The purpose is to try to find errors, missing requirements or unintended behavior by execution of the actual software and comparing the results against the expected outcome. IEEE has defined a standard for software testing. The following sections contain a summary of some relevant concepts and terminologies [25].

3.4.1 Test methods

Testing can be split into different methods depending on the amount of information and access one has to the system.

Black box testing

Black box testing is when testing is done without knowledge about the internals of a program and focuses purely on the functionality of the software. The tester provides an input and observes the output generated by the software, and no consideration is given to the internals of the software. In simple terms, it is testing as a user of the software [25, p. 46].

White box testing

White box testing is the testing of the inner functionality of a software. It requires information about the internal structure of the software, and focuses on design, usability and security. It is used to find internal security holes, verify error handling, test the flow of inputs though the code and the functionality of conditional structures. In simple terms, it is testing as a developer of the software [25, p. 46].

Gray box testing

Gray box testing is a combination of white and black box testing, where some limited information about the software is known. It is used to find errors that occur by improper use of the software. In simple terms, it is testing as a user with access to the internals of the software.

3.4.2 Test Levels

Testing can also be separated into different levels depending on how much of the software is tested at the same time. The amount of testing done is illustrated in figure 3.4, where automated testing is mostly done at the lower levels (unit tests) and user-driven testing is more common at the higher levels (system tests).

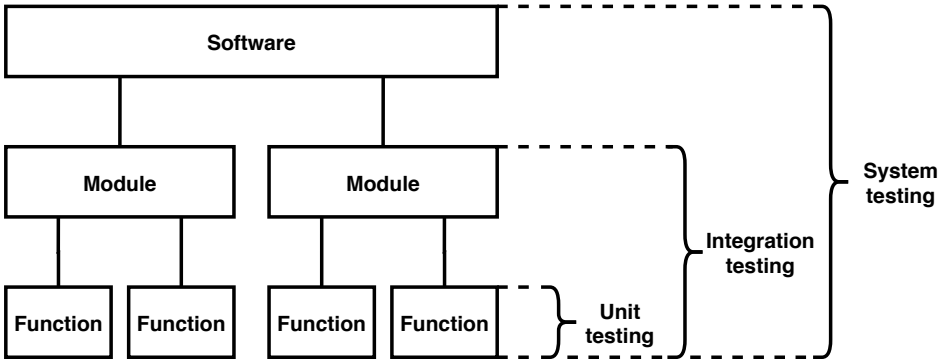

Figure 3.4: Caption

The main reason for separating the tests into different levels is to make sure that the individual functions work as intended before testing a module, and that the individual modules work as intended before testing the system as a whole. By proceeding directly to full scale system tests, the chances of errors occurring are higher. These errors would also by nature be harder to detect during large scale testing.

Unit testing

Unit testing, also called component testing, is the basic testing of some blocks of code or a specific function, referred to as an unit, to see if it functions correctly when run independently. This ensures the basic functionality of the unit is in place, and is the simplest form for testing. This can be done as automated testing, as unit testing is ideally done for each individual unit or function of code written [25, p. 8]. Unit testing is usually performed without any external dependencies, meaning that inputs and special cases are mocked up.

Integration testing

Integration testing is a step above unit testing, as this tests the functionality of modules. This differs from unit testing in that it test the functionality of the software when the units of code are combined. This is important, as functions can be functional individually, but the flow from outputs to inputs of different functions also need to be compatible [25, p. 9].

System testing

System testing is the end-to-end testing of the software, and is done to test the completeness of the software system. These tests focus on the key requirements and functionality, which differs from the code-focus of the unit and integration tests. System testing is used to verify that the software is functioning as expected and therefore are built after specifications [25, p. 10].

One way to perform system testing is by conducting acceptance tests. Acceptance testing is closely related to system testing, but is usually done in cooperation with the customer of the software before final delivery. This is a user-driven testing that verifies that the software is functioning as requested, and that it is sufficiently user friendly [25, p. 11].

3.5 nRF Connect SDK

The nRF Connect SDK (NCS) is a software development kit for Nordic Semiconductor's cellular IoT devices. It focuses on providing support for Nordic's key features which are Bluetooth Low Energy, Thread, Zigbee and Bluetooth Mesh stacks. It also provides various sample applications, reference implementations and drivers for Nordic's devices and Development Kits (DKs).

3.5.1 Development kit nRF9160

The basis of the AutroLink v1 prototype was the nRF9160 development kit from Nordic Semiconductor. As the AutroLink v2 is to be a continuation of the previous iteration, it too will be based on this development kit. The DK contains two microcontrollers made by Nordic Semiconductor. The main microcontroller is the nRF91 System in Package (SiP). It contains an ARM Cortex M33 processor and an integrated modem capable of connecting with the LTE-M and NB-IoT cellular networks. There is also a nRF52840 System on Chip (SoC) on the DK. This microcontroller functions as a board controller. Beyond this, the development kit has exposed GPIO enabling it to connect with external hardware modules. These can be controlled and communicated with through widely used protocols such as Universal Asynchronous Receiver/Transmitter (UART), Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I2C).

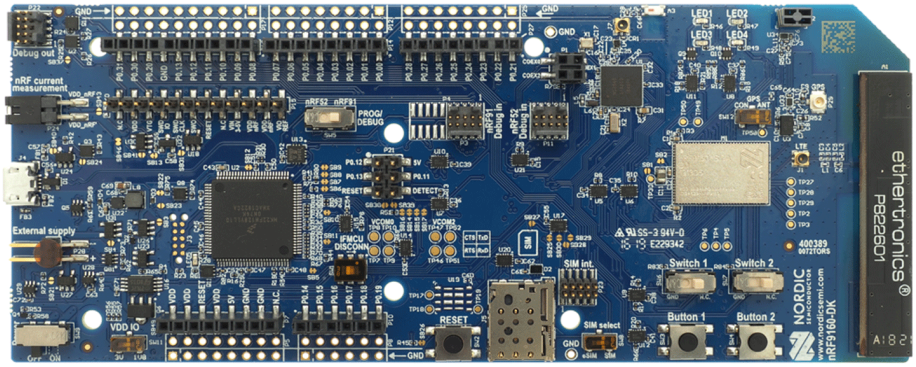

Figure 3.5: The nRF9160 DK PCA-10090 PCB

3.5.2 Zephyr Real-time operating system

Zephyr is an open source micro-kernel Real-Time Operating System (RTOS) designed to be run on Microcontroller Units (MCUs) and embedded hardware. The operating system is developed and maintained by WindRiver, and it is based on their micro-kernel profile for VxWorks. The main focus of Zephyr is IoT applications. To aid with this it has native support for many relevant protocols and standards such as Bluetooth LE, Wi-Fi, MQTT. It supports many different platforms, one of which is the nRF9160 [26].

The Zephyr OS supports kernel-services such as [27]:

- Multi-threading
- Interrupt-services
- Inter-thread synchronization
- Inter-thread communication
- Priority and scheduling

3.5.3 Managing multiple repositories with West

NCS is a multiple repository environment, consisting of the following repositories:

- **nrf:** A repository containing source-code additions, making Nordic Semiconductor's cellular IoT devices compatible with the Zephyr RTOS and MCUboot
- **nrfxlib:** A repository that contains RTOS-independent libraries and modules to be used with Nordic Semiconductor SoCs
- **MCUboot:** A secure and versatile bootloader for 32-bit microcontrollers

- **Zephyr project:** A scalable RTOS built with security in mind, suitable for MCUs and embedded hardware

In order to keep track of which versions of the different repositories are compatible with each other, Nordic recommends using Zephyr's meta-tool West [28]. At certain points during the developments, the compatible commits are assigned a unique identifier called a "tag". How West keeps track of the different commits in the three repositories is visualized in Figure 3.6. In this figure, the gray dots represent a commit, a version of the code in the repository. The tags can be seen as the commits along the top row. Each of these point to a specific commit in each of the three repositories. When working with NCS it is possible to use West to jump between the different tags.

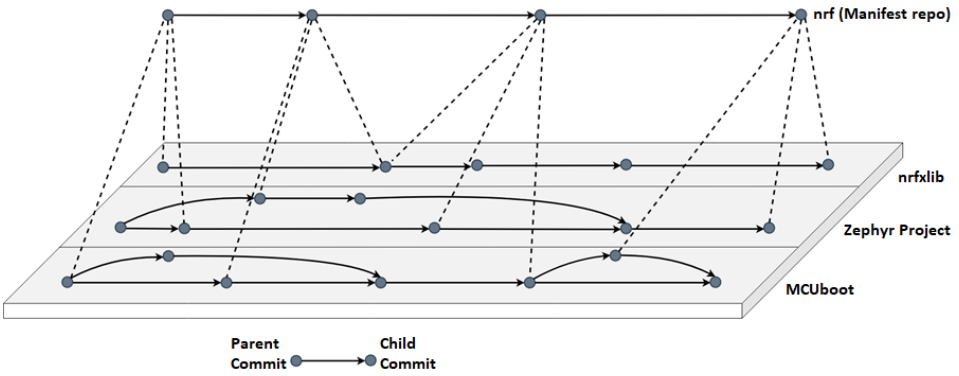

Figure 3.6: Multiple repository management using West, illustration courtesy of Nordic Semiconductor, edited for readability

3.5.4 Version control with Github

NCS, as well as the software development performed as part of this thesis, utilizes Github for version control. Github is a web-based versioning tool built upon Git, an open-source version control system. It allows the project-code to be available anywhere, and provide a backup if data is lost. It also allows for collaboration by tracking overlapping code changes from different developers, and giving the option to merge these changes before adding them to the main project. If one wants to test different ways of designing a feature, it's possible to split the project development into separate tracks, branches, and later choose which to keep and merge back into the main project [29].

Chapter 4

Development of specification

In this chapter, the information that has been gathered so far will be analyzed in order to arrive at a specification for the AutroLink v2 prototype. The starting point of the analysis is the AutroLink v1 prototype that was designed, built and tested during the specialization project preceding this master's thesis. The analysis focuses on what kind of functionality that needs to introduced in order for the prototype to be helpful during service and commissioning. The goal of the preliminary analysis is to locate the opportunities and restrictions of the AutroSafe system, and propose functionality. This leads up to the requirements for the new prototype. From here, a structured analysis will be performed for modularizing the system and arrive at a specification.

4.1 Opportunities for functionality

During the commissioning routine the AS-2000 software is used to verify the detection loops. A computer running the software is then either connected directly to the module stack of the corresponding panel or to the detection loop utilizing the WAS-2000. An important part of this verification process is to check if the detectors have been mounted correctly and in the correct location. This is usually done by turning on the LED of a detector and locating them within the facility. Plans and drawings can then be consulted to check if the detector is in the correct location. Since turning on more than one LED can lead to confusion, only one is turned on at a time. This means that the service engineer has to move to it's location in the installation and back again to the computer for each detector. This either involves a lot of unnecessary walking or more service engineers present attending this one task.

By introducing a wireless gateway, the detection loop verification procedure can be separated into two parts. The first part is where AS-2000 is used to verify that the topology of the detection loop is correct. The second part is where the AutroLink v2 is used to allow the service engineer to move around while controlling the LED's remotely, using a smartphone, a tablet or a portable computer of some kind. This would simplify the process by reducing the number of people present and/or reducing the time required to perform these checks. For this to be possible, the AutroLink v2 needs to support two-way communication with the AutroSafe system.

During service there is one point where remote access could be helpful. That is mainly the ability to bring in expert knowledge in situations where the standard service routines are unable to unveil the problems. An expert having the opportu-

nity to communicate directly with the module stack, and through it, the individual detection loops and loop units, could identify errors that are otherwise hard to detect. Here, the communication's level of abstraction matters. The less abstracted the communication is, the more information it contains. During service it would be most beneficial if the messages sent to and from the additional MQTT clients were on the lowest level possible, which in this context is sending AL_Com+ messages directly. This way, the operator will be able to see the flow control messages and the directives passed between the AutroLink v2 prototype and the units in the module stack. Additionally, since the AL_Com+ messages are usually interpreted as hexadecimal numbers, they should be converted to readable characters in order to make them easier to read.

4.2 Autronica's requested functionality

Through discussing with the engineers of Autronica's R&D department, it was decided to add certain functionalities in the AutroLink v2 prototype. The operator should be able to not only use AL_Com+, but also AL_Com to communicate with the module stack through the AutroLink v2. This will enable the AutroLink v2 prototype to be tested in conjunction with prototype software that is being developed. The software in question is able to simulate different parts of the AutroSafe system. R&D will use the software to simulate a panel with a module stack containing a loop driver. Furthermore, it will be able to connect with AutroMaster ISEMS. This means that the software will send AL_Com messages to the AutroLink v2, as if it was communicating directly with a detection loop. The AutroLink v2 prototype should therefore be able to convert the messages coming from the MQTT server from AL_Com to AL_Com+ before conveying the messages to the physical module stack and loop driver. The software module also expects regular AL_Com messages in return, so the AutroLink v2 should be able to convert the AL_Com+ messages coming from the physical AutroSafe system back to AL_Com messages before sending them to the MQTT server. This way the prototype and the physical module stack would be transparent. A thing to note is that the software, as well as the MQTT broker hosted by Autronica does not support TLS nor any form of mutual authentication.

The ability to convert back and forth between AL_Com and AL_Com+ could be useful in other situations. In some cases the extra information in AL_Com+ might be unnecessary. One example of this is if the maintenance is to be performed at a detection loop and not the module stack. There could also be other cases where the flow control messages present in AL_Com+, as well as the information from the header byte of the AL_Com+ message are unnecessary. Therefore, MQTT clients should have the opportunity to communicate using AL_Com.

4.3 Interfacing with the AutroSafe system

Following the analysis of the AutroSafe system and the requirements of the new prototype, it is seen that the method of system integration of the AutroLink v1 prototype cannot be used for the AutroLink v2 prototype. The new functionalities all necessitate two-way communication with the AutroSafe system. Looking back at the report written during the specialization project, four methods of integration were discussed. While the focus during the specialization project was data extraction, some of the methods allowed for two-way communication [1]. The following paragraphs re-evaluate a selection of these taking the new requirements and information into account.

The first suggestion was to connect the prototype directly to the detection loop, as illustrated in Figure 4.1. This would allow the AutroLink to connect with every loop unit on the detection loop. It would also allow the prototype to be connected anywhere along the loop, not only next to a panel. However, a major downside to this solution is that the detection loop needs to be raised in order for the communication to be able to take place. Another downside is the fact that the prototype would not be able to communicate with the modules in the module stack.

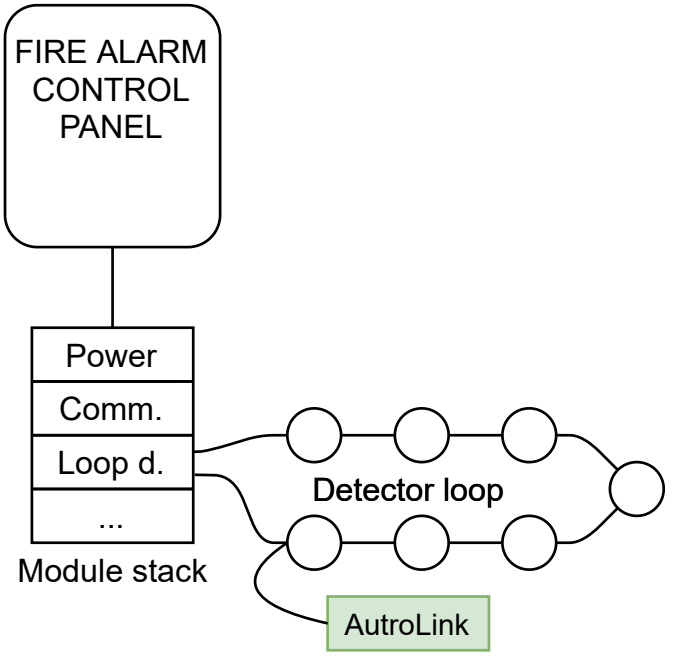

Figure 4.1: The first implementation mentioned in the specialization project report [1]

The second suggestion was to disconnect the panel from the module stack, and have the prototype connect with the module stack instead. This would connect the prototype with the module stack through the communication module, making it able to communicate with every unit in the module stack and subsequently every detection unit in the connected detection loops. If disconnecting the entire module stackmodule stack is undesirable, a spare module stack consisting of a power supply module, communication module and loop driver could be brought along the AutroLink. A single detection loop could the be disconnected from the AutroSafe system, connecting it with the AutroLink v2 through the spare module stack. The WAS-2000 service suitcase could also be used for this task.

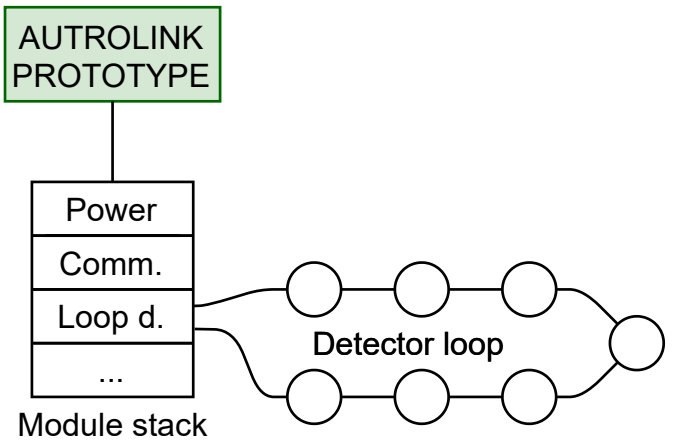

Figure 4.2: The second implementation mentioned in the specialization project report [1]

Therefore, it seems that by connecting the AutroLink v2 prototype with the communication module of the module stack we get most of the upsides and few of the downsides. From a both a safety and security standpoint, this implementation is better than letting the gateway access the entire system. With regards to safety, only one module stack is disconnected from the system. The rest of the system can function as normal, albeit with errors and faults present until the service is finished. With regards to security, only one module stack is exposed to the Internet. The module stack that gets disconnected from the AutroSafe system will not function as normal during the service. This means that eventual fires will not be detected and no alarms will be raised. However, the same is true when AS-2000 is used to communicate with the loop, as is routine during service and commissioning.

4.4 Requirements

To summarize; the AutroLink prototype can benefit Autronica's service and commissioning routines by allowing operators to remotely connect with the AutroSafe system's module stack. Two-way communication between the AutroLink v2 and the module stack opens up for the opportunity for operators to work remotely and communicate with the module stack units and detection loops. The ability to chose between communicating with AL_Com and AL_Com+ opens up certain possibilities, one of them being interfacing with Autronica's prototype simulation software and AutroMaster. This has been condensed into the requirements listed below.

- Remote connection to the AutroSafe system
- Operator must receive directives in the form of readable text from the AutroSafe system
- Operator must be able to write directives using readable text and send it to the AutroSafe system
- Operator can choose between using AL_Com and AL_Com+ directives remotely

4.5 Structured analysis of AutroLink v2

In order to get a better understanding of the requirements for the prototype and how they affect the design, performing a structured analysis was part of the tasks described in the problem description. This can help visualize how the prototype's functionality, hardware and software can be separated into modules, as well as which module ensures which requirement can be met.

The analysis is conducted in the following manner:

- The system and it's environment is described, often using a context diagram as a visual aid. The context diagram shows how the system is connected with other systems and which information they exchange.
- The system's functionality can then be separated into individual modules. This is shown using a diagram called an inner analysis of the system. These modules can represent hardware, software or a combination of the two. The diagram also shows how the internal modules are interconnected. For this particular project the inner analysis shows how the prototype's software was modularized. An analysis that addresses hardware or a combination of hardware and software would differentiate greatly.
- The individual internal modules can be separated into smaller modules, like the inner analysis showed the modularization of the system from the context diagram. This process can be repeated until it is feasible to conclude with a specification. For this project, the analysis stops at the second level of modularization.

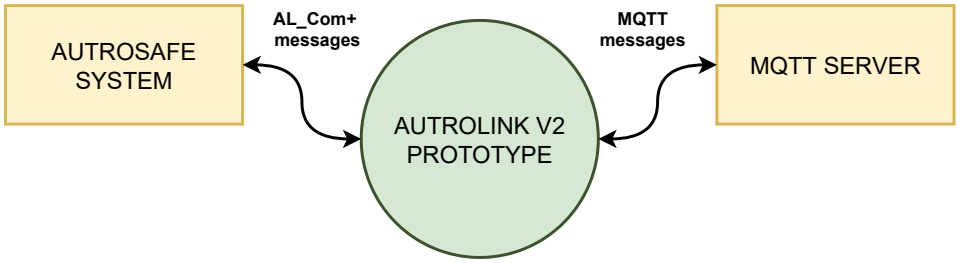

Figure 4.3: Context diagram

The starting point of the analysis conducted here is the AutroLink v1 prototype from the specialization project, with some slight modifications. Temporary installation while there are service engineers or other trained personnel nearby. Prototype can be placed inside cabinet to shield from potential dust and moisture exposure. Through the initial analysis conducted in Section 4.1, it was found that the new prototype would need two-way communication with the AutroSafe system in order to fulfill the needed service functionality. The AutroLink v2 prototype needs to be able to interface with two separate systems. These external systems provide certain limitations that the prototype needs to be built around. In particular, the kind of information that gets exchanged, timing restrictions and how it gets transported.

The first external system is the AutroSafe system, or more specifically, a module stack in the AutroSafe system. This is visualized in Figure 4.3 as the yellow box to the left. It needs to support both transmitting and receiving AL_Com+ messages at a transfer rate of 9600 baud. From the AutroSafe system's side, these messages get converted from TTL to RS-232. On the AutroLink's side the messages need to be converted back from RS-232 to a TTL-level suitable for the nRF9160 DK. The AL_Com+ communication imposes certain timing and sequencing restrictions on the prototype, as to make the communication function as normal. Additionally, the modules of the module stack will not initiate the AL_Com+ communication, but rather respond when necessary.

The second external system it needs to interface with is the MQTT server and the additional clients, which can be seen as the yellow box to the right in Figure 4.3. In order to communicate with this server the prototype needs a functional implementation of the MQTT protocol and it needs to run an MQTT client. Furthermore, it needs to establish and maintain a connection with the server over the Internet. This external system does not impose any particular timing restrictions on the prototype. However, the messages from the MQTT server can arrive at any time.

4.5.1 Inner analysis

Following the context diagram in Figure 4.3, it is seen that the AutroLink v2 prototype needs to communicate with two separate external systems. Since the messages from coming from the MQTT broker can arrive at any given moment, they can be regarded as aperiodic events that need to be handled. On the other side, the communication with the AutroSafe equipment is generally in the form of responses to requests from the prototype during service. One way to modularize the program is to separate these two distinct tasks into separate two modules, as seen in Figure 4.4.

Another requirement for the system is to convert the AL_Com+ messages into readable characters. This functionality was assigned to the AutroSafe interface module. This means that the information exchanged between it and the MQTT client module are the readable AL_Com+ messages.

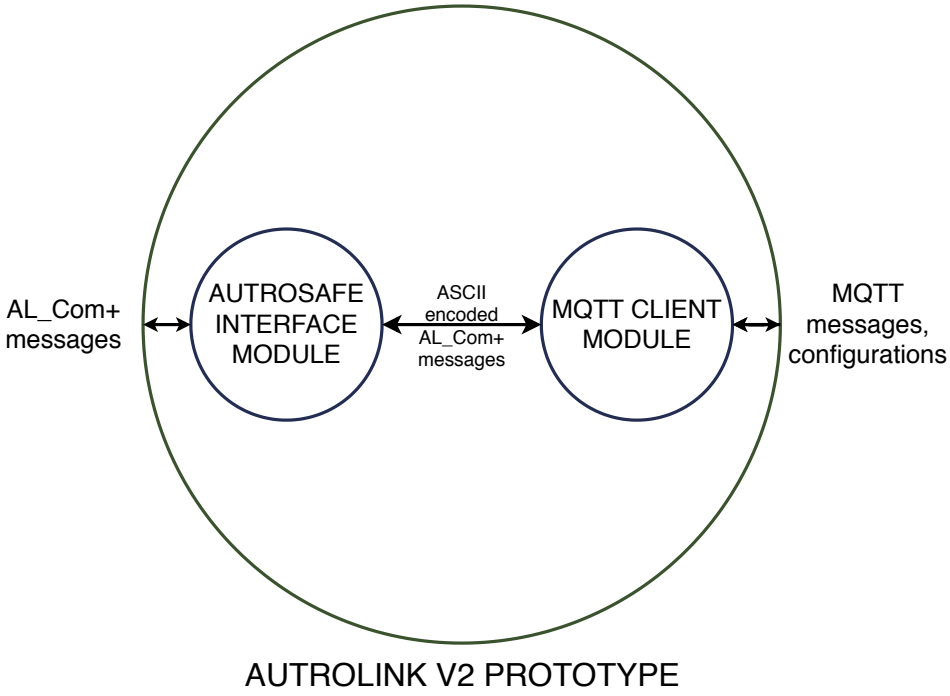

Figure 4.4: Inner analysis of the AutroLink v2's software

4.5.2 AutroSafe interface

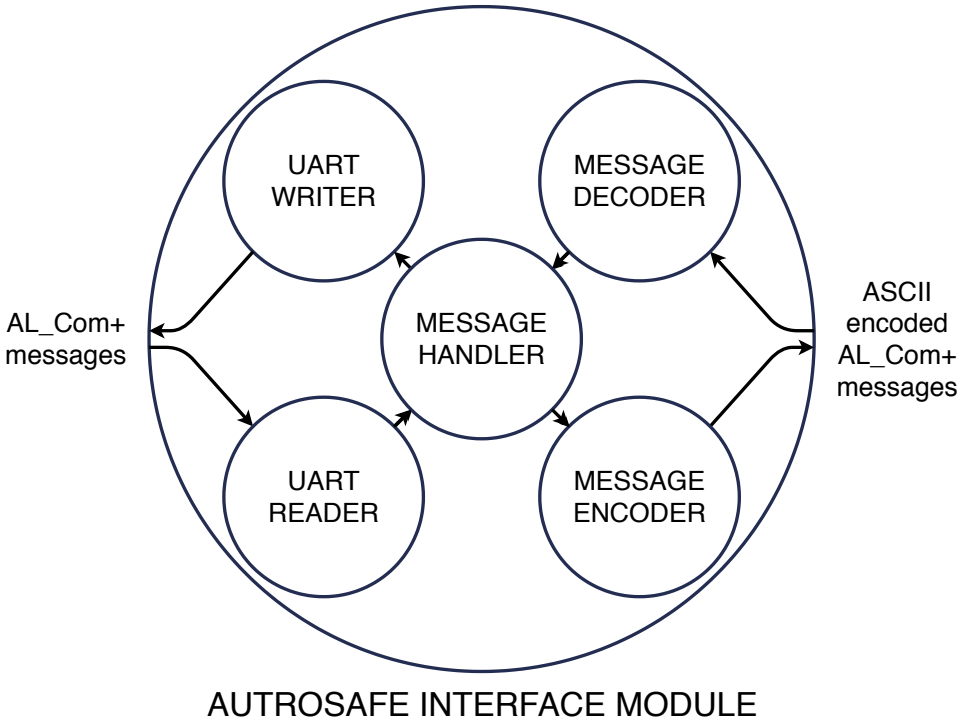

Figure 4.5: The AutroSafe interface module

In:

- AL_Com+ responses from the AutroSafe system
- Readable AL_Com+ messages from the MQTT client module

Out:

- AL_Com+ messages to the AutroSafe system
- Readable AL_Com+ messages to the MQTT client module

The functionalities of the AutroSafe interface module is further separated into submodules, as seen in Figure 4.5. The message decoder and encoder modules are responsible for converting the messages between readable characters for the operator and hexadecimal values for the AutroSafe system. The message decoder is responsible for converting the AL_Com+ messages before these get passed to the message handler. The message encoder will convert the raw AL_Com+ message to a readable format before passing it to the MQTT client module. The message handler controls the communication with the AutroSafe system and is responsible for deciding when to send the AL_Com+ messages to to the AutroSafe system and

when to listen for responses, as well as which responses get passed to the message encoder. In addition to these tasks, these three sub-modules should check the local configuration of the prototype to determine whether the prototype is operating in "service mode" or "transparent mode".

If it is operating in "transparent mode", the messages coming from the MQTT client module will be readable AL_Com messages, as opposed to AL_Com+ messages. The message decoder then also needs to convert the AL_Com messages to AL_Com+ before these get passed along to be sent to the module stack. Following the lines of the decoder, the encoder will have to be able to convert the AL_Com+ messages coming from the AutroSafe system into AL_Com messages before passing them on.

The UART writer and reader are responsible for reading and writing the messages to the AutroSafe system. In case the AutroSafe system responds while the prototype is preoccupied with something else, the AL_Com+ message should get stored in a buffer for incoming messages.

4.5.3 MQTT client

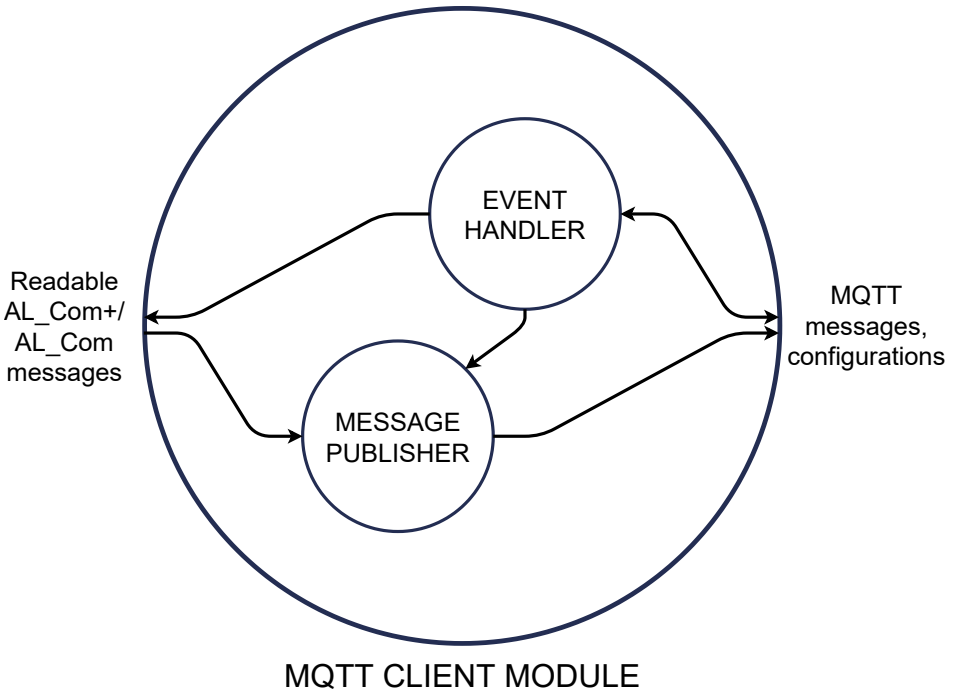

Figure 4.6: The MQTT client module

In:

- MQTT messages from the MQTT server
- Readable AL_Com+ messages from the AutroSafe interface module

Out:

- MQTT publish messages to the MQTT server
- Readable AL_Com+ messages to the AutroSafe interface module

The "event handler", as seen in Figure 4.6, submodule has the main responsibility of handling MQTT events. Upon startup it should establish a connection with the specified MQTT server. In order to do this, it needs to send and receive messages to handle the necessary handshakes and procedures. Upon a successful connection, it should subscribe to the specified topics. If a message arrives on a topic the AutroLink v2 prototype is subscribed to, the event handler should route the message accordingly. Messages requesting changes to local configuration should result in the relevant global variables being changed correctly. Messages meant for communicating with the AutroSafe equipment should be routed to the AutroSafe interface module. Some messages sent by the operator might necessitate an answer from the AutroLink v2 prototype. The event handler should therefore have the possibility of offloading certain message publishing tasks to the "message publisher" submodule.

The "message publisher" has the main responsibility of publishing MQTT messages that do not originate from an MQTT event, such as the messages coming from the AutroSafe system. The message publisher should receive information from the event handler and the AutroSafe interface module. These should be published to the specified topic as an MQTT message.

4.6 System specifications

When summarizing the functionalities of the modules in the previous sections the following specifications are found.

1. Connect to and communicate with a specified MQTT server
2. Subscribe to given topics and send/receive information through these topics
3. Process the received messages
4. Be able to translate between binary values and readable text, and between Autronica's proprietary protocols
5. Physically interface with the 10 pin latch contact on the BSL-310 Communication module
6. Be able to read and write UART signals with a minimum speed of 9600 bps

Chapter 5

Design and implementation

This chapter documents the proposed design to meet the requirements for the AutroLink v2 prototype. How this design was implemented is also documented.

5.1 Development environment

During the project report, the chosen development environment was a laptop running Ubuntu 18.04. For this thesis the development was moved to another computer running Windows 10. In broad strokes, setting up the development environment for the two projects is similar.

The nRF Connect Software Development Kit (SDK) was set up using the nRF Connect application [30]. This guides you through installing the necessary tools and programs in order to work with the SDK, as well as compiling, building and flashing applications for the DK.

Since the AutroLink v1 prototype was developed using NCS tag v1.2.0, the same tag was used for the development of the updated prototype's firmware. A Github repository was created as a fork of this tag. This made it so that the prototype's firmware being developed within Nordic Semiconductor's SDK also had version control.

Unlike Linux, Windows does not have a built in terminal command that enables the reading of serial data. There are software tools available such as PuTTY [31], which are commonly used used to perform this task. However, for this project it was opted to write a Powershell script that performed this task in a terminal. The script can be found as part of Appendix A. In the cases where it was necessary to write serial data from the computer, PuTTY was used.

5.1.1 Programming practices

The code was written using the case style snake_case, where function and variable names use lower case letters separated by underscore. An effort was made to ensure that the code was readable. Descriptive names were given to variables and functions, so that comments in the code were generally unnecessary. A code formatter for C was used to keep the code tidy.

5.2 System overview

The proposed design for the prototype is to make it interface with the module stack through the communication module. This means it will connect to the 10-pin latch contact to both transmit and receive AL_Com+ messages. As with the AutroLink v1 prototype, the nRF9160's Low Power Wide Area Network (LPWAN) modem will be used to establish an Internet connection through the LTE-M network. An MQTT server will be used to allow clients to connect with the prototype in order to perform service and commissioning routines on the connected AutroSafe system. It should be possible to communicate with the connected AutroSafe system using AL_Com+ and AL_Com. An overview of the proposed design can be seen in Figure 5.1.

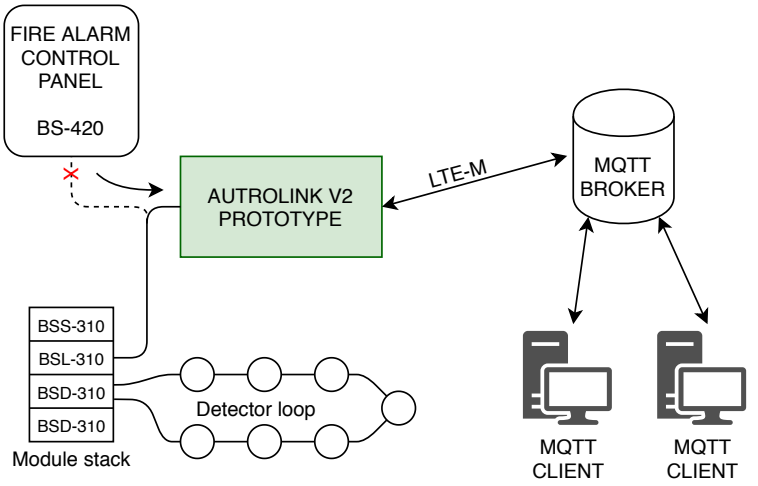

Figure 5.1: Concept illustration of the proposed solution for the AutroLink v2

Here, it is possible to use the same MAX3232 circuit that was used in the AutroLink v1 prototype. However, changes must be made so that the TX, RTS and CTS signals are also usable, since the only signal line that was tested in the previous prototype was the RX signal. The proposed solution is therefore to use most of the AutroLink prototype's hardware as is, and only make minor changes in order to meet the requirements of the new prototype.

5.2.1 Changes to the hardware

Few hardware changes were necessary in order for the AutroLink v2 to be able to meet the new requirements. Two additional wires were connected from the nRF9160 DK board's GPIO to the MAX3232 circuit's CTS and RTS lines. The wires were added added to the construction by selecting two free GPIO pins, and are represented by the green and yellow wires.

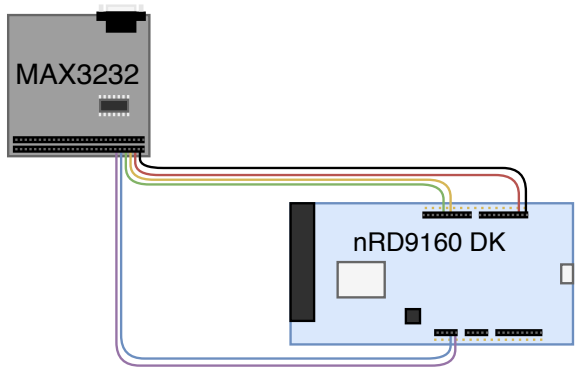

Figure 5.2: Illustration of the nRF9160 PCA-10090 connected with the MAX3232 circuit

5.2.2 System design choices

In order for the operator to be able to start the service work as soon as possible after connecting, the AutroLink v2 should perform the addressing of the module stack. This should happen automatically after the prototype is connected to the system and powered on. The operator can then either send identification directives to each module in the module stack or base the communication on existing schematics and drawings of the system. If there is something wrong with the module stack, e.g. this is the reason for the service, this automatic addressing should terminate in a way that still makes it possible for the operator to communicate with the AutroLink v2 prototype and the AutroSafe system.

In order for the operator to be able to send directives to the individual detection units along the detection loop, the loop has to have been raised beforehand. If the required service work was to send directives to a specific detector, it would be practical if the AutroLink v2 prototype performed the loop raising procedure automatically upon connecting to the system. There are however arguments for why this procedure should be done manually instead. Raising the detection loop requires handling of several special cases and can be time-consuming in larger installations. The prototype would also need to continuously send messages about which AL_Com+ messages it sends to the AutroSafe system, and which messages it receives. The loop raising feature is already a part of the AS-2000 software, which logically should already have been attempted before using the AutroLink v2 prototype. Additionally, performing this task manually inherently enables the operator to see every step of the communication. Therefore this procedure will be done manually in this version of the firmware. Moreover, if a loop driver already has a raised loop, the loop will stay raised when connecting the AutroLink v2 to the module stack and performing the module stack addressing procedure, as stated in Section 2.8.4.

Adequate fault detection and handling is necessary when building a robust applica-

tion. For the AutroLink v2 prototype it was opted to go for a crash-only approach. As soon as the software enters an unknown state or discovers an unexpected result, it should crash and restart. This was chosen as the prototype does not need to store any important information, and only is to be used in service situations. Implementing this behavior is also a way of allowing the device to restart without the help of a user.

5.2.3 Terminologies

In order to create a better overview, certain terminologies were defined during development. Some of these are illustrated in Figure 5.3. Both AL_Com and AL_Com+ will be sent between the AutroLink v2 and the operator of the remote MQTT client. In order to not create confusion in which way a message is headed, the terms "upstream" and "downstream" were created. An upstream message is a message that originates either from the AutroSafe system or the AutroLink v2 prototype and is headed towards the operator. A downstream message is a message that is sent from the operators MQTT client, and is meant either for the AutroSafe system or the AutroLink v2. There are multiple MQTT clients connected to the MQTT server. One of the clients is specified to be a module of the AutroLink v2 functionality, which will be referred to as the "MQTT client module", as in Section 4.5. Another significant MQTT client is the one used by the operator to send downstream messages to the AutroLink v2 and AutroSafe system. This will be referred to as the "operators MQTT client". The operator is a person who is knowledgeable about the AutroSafe system, the service and commissioning routines and is able to read and understand both AL_Com and AL_Com+ messages.

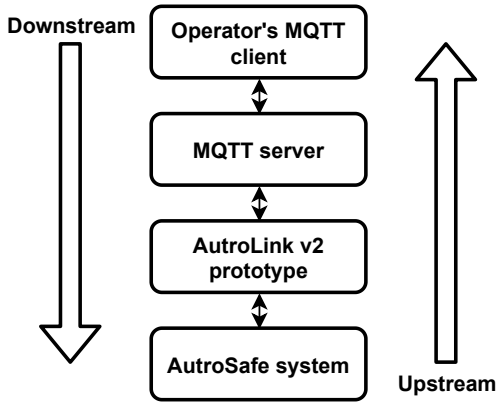

Figure 5.3: Figure illustrating some of the terminology defined during development

5.3 MQTT client and message handler

The MQTT module of the software is a continuation of the module used during the specialization project [1], which in turn is based off of Nordic Semiconductor's code sample [32]. This sample provides an MQTT client built using Zephyr's MQTT library [15], that is able to run on the nRF9160 SiP. The sample code has the client subscribe and listen to one MQTT topic. If it receives a message on this topic, the client publishes the same message to another topic. The code was altered to not echo the incoming messages to another MQTT topic, but rather process them consecutively. The subscribe function is set up to subscribe to what it perceives as a list of topic names. As default, the sample only specifies a single topic. New topics were therefore added to the subscribe function as new elements. These were specified and named in the prj.conf file as seen below, along with the rest of the necessary changes to the MQTT client.

```
CONFIG_MQTT_PUB_TOPIC = "ALK_v2/panel-1/Upstream"
CONFIG_MQTT_SUB_TOPIC = "ALK_v2/panel-1/Downstream"
CONFIG_MQTT_SUB_TOPIC_2 = "ALK_v2/panel-1/config"
CONFIG_MQTT_CLIENT_ID = "AutroLink_v2"
CONFIG_MQTT_BROKER_HOSTNAME = "mqtt.eclipse.org"
CONFIG_MQTT_BROKER_PORT = 1883
```

Listing 5.1: Contents from the prj.conf file

5.3.1 Handling incoming messages

The incoming messages get analyzed to see whether it is a message containing configuration changes. If this is the case, the changes to the local configuration are made. If not, the message gets passed to the AutroSafe interface module through a First-in First-out (FIFO) queue. Here each element that gets passed contains a pointer to the message stored in memory and an 8-bit integer value of the message length. The longest AL_Com+ messages are 17 bytes. When converted to readable symbols they take up twice as much storage space, since American Standard Code for Information Interchange (ASCII) uses one byte to encode each character. This results in the largest possible readable AL_Com+ message being 34 bytes large. This adds up to 37 bytes per message when including the the 8-bit integer for length and a 2-byte pointer used by the kernel to reference the FIFO element [33]. Calculating the the exact number of messages that can be stored in this queue is insignificant, as the queue is dynamically allocated from the calling threads resource pool. The event of the queue becoming "full" would likely be an indication of something not working as intended. In this case the nRF9160 would perform a restart as the code that allocates space for the FIFO queue would run out of available memory, adhering to the crash-only mentality. Outgoing (upstream) messages get stored in a message buffer until they have been transmitted. In the case of the message having a QoS of 1 or 2, it stays in the message buffer until the message gets acknowledged by the server. The outgoing message buffer of the MQTT client is by default set to 128 bytes. This means that that the worst-case

number of possible stored messages in this buffer is $\lfloor \frac{128byte}{35byte} \rfloor = 3$, and the best case is $\lfloor \frac{128byte}{3byte} \rfloor = 42$. If necessary, the buffer size can be increased.

While writing and evaluating this module it was discovered that it could in some cases lose connection to the MQTT server, in which case the thread would terminate without causing the prototype to restart. This was due to how the MQTT event handler responded to certain return codes from the MQTT CONNACK messages (Connection acknowledged), where the return code "0" means the connection was accepted and any other return code signifies some sort of error [14]. The event handler was changed in order to adhere to the crash-only mentality, by including the `sys_reboot()` function in the case where the return code was not 0. This function reboots the nRF9160 DK, causing it to automatically start up again.

5.3.2 Message publishing

The MQTT client module needs to be able to publish messages at other times than only at an MQTT event. One way to achieve this is to handle it outside of the MQTT event handler. This was solved by offloading the publishing of the messages to a workqueue thread. Zephyr provides workqueues, which are kernel objects consisting of a queue of work items and a thread that executes the work [34]. As part of the initialization of the MQTT client module, 16 work items get initialized. This leads to Zephyr starting up it's system workqueue [35]. The worqueue thread functions as the message publisher submodule in Figure 4.6.

A function for putting the upstream messages into the workqueue was written. As with the FIFO queue that passes the downstream messages to the AutroSafe interface module, this function takes in both a pointer to an array and an integer for the message length. Once a work item has been submitted it's important that the contents are not altered until the work item has been processed by the workqueue thread. This function therefore cycles through the 16 work items, giving the workqueue thread time to finish processing. This was mostly done as a safety feature, as the workqueue thread's priority is higher than the other threads in this application. This function used a static integer to keep track of which of the workqueue threads should receive the next message.

5.4 AutroSafe interface module

The thread that runs the AutroSafe interface module is responsible for conveying messages to and from the AutroSafe system. The code from the specialization project that handled the reading of AL_Com+ messages differed greatly from the needed functionality. This code was therefore scrapped and written from the ground up.

When analyzing the kind of tasks this software module needs to handle it is seen that they largely consist of aperiodic tasks. This is when the MQTT client module sends a message to this module to be sent to the module stack. From the other system that the prototype is interfacing with, the AutroSafe's module stack, the messages will be in the form of a response. This means that while the AutroLink v2 is being used, the same sequence of events will be repeated until the maintenance work is over. First, the remote client will send an AL_Com+ directive to the AutroLink v2. Then, this message will be converted from readable characters to hexadecimal values before being sent to the module stack. The AutroLink v2 prototype will then wait for a response from the module stack. This response will be converted from hexadecimal values to readable characters, and get sent to the MQTT server. When the AutroLink v2 prototype is to operate in transparent mode, the operator sends AL_Com directive instead. In this case, the encoder and decoder also need to convert the downstream message to an AL_Com+ directive, and the upstream message to an AL_Com directive. Additionally, since the flow control messages present in the AL_Com+ protocol is not part of the AL_Com protocol, the AuttroSafe interface module also needs to handle this low level communication automatically akin to how they are treated in Section 2.8.4. Certain messages, such as directives, need to be acknowledged using the ACK flow control message. Sometimes it is necessary to use an ENTX flow control message to request the information a unit has prepared. By using what's called a "big while loop" we're able to handle these events in a sequential manner, as visualized in Figure 5.4.

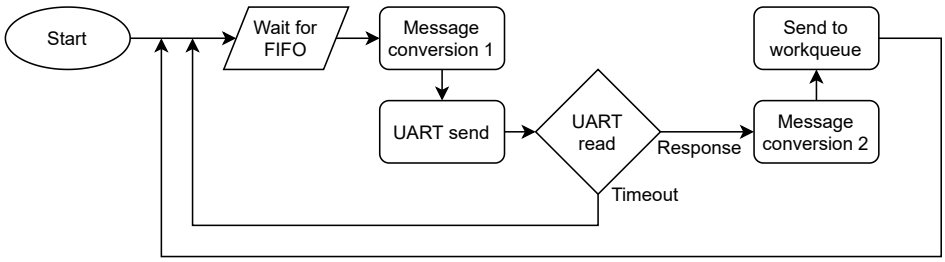

Figure 5.4: Flow chart depicting the main loop of the AutroSafe interface module thread

As the AutroSafe system is to respond to flow control messages and directives, the main loop of the AutroSafe interface module should wait for a message coming from the FIFO queue from the MQTT client thread, as shown in Figure 5.4. Once a message appears in the queue, it should be converted from readable characters to binary before getting sent to the module stack. The main loop should then wait for a response from the module stack. This can result in one of two events. The first event is that the module stack responds, in which case the message should be converted from binary to readable characters before getting offloaded to the workqueue threads. The second possible event is that the module stack does not respond. This case can be detected through the use of a timeout timer. Both events result in the AutroSafe interface module waiting on the FIFO queue from the MQTT client thread again.

Both the messages coming from and being sent to the MQTT client module requires a pointer to a character array and information about the length of the message. The same is true for the messages being picked up and sent out on the UART. In order to keep the message flow tidy as the messages get passed from function to function certain variable and structures were defined using specific names. This was done using a keyword in C called typedef [36]. The following types were defined:

```
typedef unsigned char bin_data_t;
typedef bin_data_t *bin_str_t;
typedef struct bin_msg_t {
  size_t len;
  bin_str_t data;
}bin_msg_t;

typedef char hex_data_t;
typedef hex_data_t *hex_str_t;
typedef struct hex_msg_t {
  size_t len;
  hex_str_t data;
}hex_msg_t;
```

Listing 5.2: Custom types for the datastructures used in the firmware

The naming convention used here is as follows. “Data” refers to a single byte from a message, “str” is a pointer to the first byte of a message without any information of the length, while “msg” signifies a structure containing both the aforementioned pointer and information about the message length. All references to “hex” refer to the readable versions of the AL_Com and AL_Com+ messages, while “bin” signifies the messages that are strictly hexadecimal. This naming convention was specifically requested by the engineers at Autronica’s R&D department.

5.4.1 Addressing the module stack

As stated in Section 5.2.2, this module should first perform the module stack addressing procedure. Following the descriptions in Section 2.8.3, a loop performing this procedure after the thread initialization was made. This was executed before entering the main loop of the module.

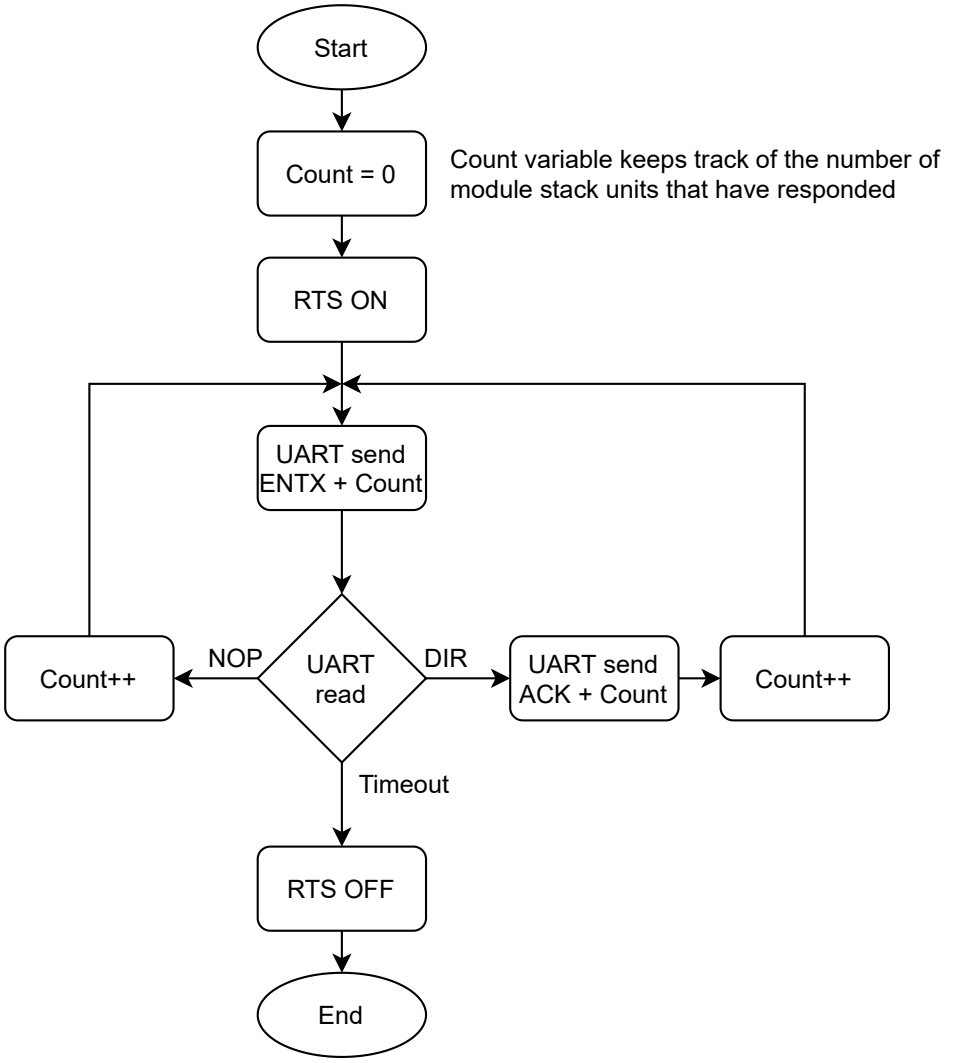

Figure 5.5: Flow chart portraying the algorithm for addressing the module stack

After the prototype powers up and finishes initializing, the thread housing the AutroSafe interface module starts communicating with the module stack. It first

activates the RTS signal line, initiating the module stack's addressing. By sending ENTX flow control messages and handling the responses from the module stack accordingly, as can be seen in Figure 5.5, the modules gain individual addresses, making it possible to communicate with each individual module. The algorithm finishes after it has sent an ENTX without receiving a response within 100ms. At this point the "count" variable will represent the number of modules in the module stack given that no messages were lost during communication.

5.4.2 Message conversion

In order to make writing and reading the AL_Com+ messages easier for the operators connecting with the AutroLink v2, the hexadecimal values should be changed to readable symbols. Using an ASCII-table, it is possible to find the numerical values for the hexadecimal numbers represented by ASCII symbols. Since the AL_Com and AL_Com+ directives are documented and represented by the hexadecimal numbering system, we need access to the numbers 0-9 and the letters a-f and/or A-F. As seen in Table 5.1, the numbers 0-9 are represented by the hexadecimal values 30-39, the upper case symbols A-f by 41-46 and the lower case symbols a-f by 61-66. This also means that when the directives are written using readable symbols, they take up twice as much space as when written with hexadecimal values.

Hex value	24	47	6E	21	...
ASCII symbol	#	G	n	!	

ASCII symbol	2	4	4	7	6	E	2	1	...
Hex value	32	34	34	37	36	45	32	31	

Figure 5.6: Above is an example of a message written in hexadecimal values converted to readable symbols. Below is the same message written with readable symbols converted to hexadecimal values

Symbol	Hex	Dec
0	30	48
1	31	49
2	32	50
3	33	51
4	34	52
5	35	53
6	36	54
7	37	55
8	38	56
9	39	57

Symbol	Hex	Dec
A	41	65
B	42	66
C	43	67
D	44	68
E	45	69
F	46	70

Symbol	Hex	Dec
a	61	97
b	62	98
c	63	99
d	64	100
e	65	101
f	66	102

Table 5.1: Readable symbols of the hexadecimal numbers 0-F and their corresponding hexadecimal and decimal values

Two functions were written to handle the conversion of messages. The first function converts the upstream AL_Com+ messages from hexadecimal values to ASCII, and was called `bin_to_hex()`. The second function does the opposite for the downstream messages, and was called `hex_to_bin()`. Both functions make use of the defined types mentioned in Section 5.4.

Downstream conversion

The function that converts the downstream messages takes a structure of the type `hex_msg_t` as an argument. This argument contains both a pointer to the first byte of the message array and an integer value of the message length. This way, the function is able to locate where the message is stored in memory and read the correct number of bytes. Since these messages consist of readable characters, each symbol takes up a whole byte as opposed to a nibble, such as a hexadecimal character does. The function iterates through the array and converts the readable characters to hexadecimal values using the same information as in Table 5.1. Afterwards, it performs bit-shifting operations in order to compress the information from every two bytes into one byte, as shown in Figure 5.7. The top row labeled "Symbol" represents the readable symbols of the downstream message before conversion. The line below are the hexadecimal values of these readable symbols. The hexadecimal number A corresponds to the number 10 in the decimal number system. By subtracting 55 we have successfully converted this specific byte. Moving downwards to the second half of the figure, the now hexadecimal A gets bit-shifted four bit spaces to the left and added together with the following byte. The function then returns the converted message using a pointer to a structure of the type `bin_msg_t`.

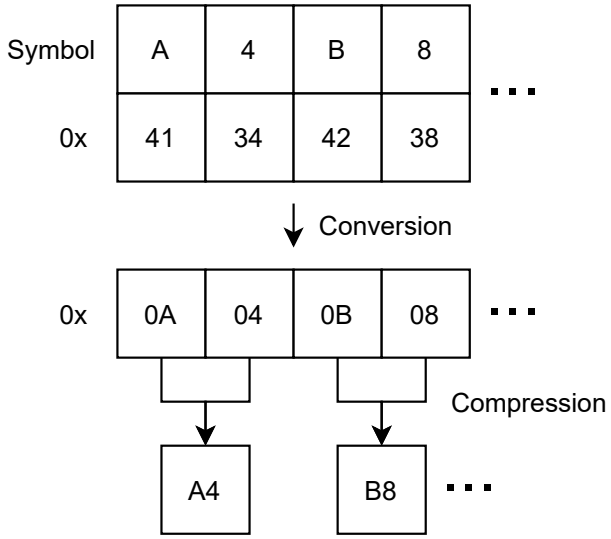

Figure 5.7: The individual bytes of a downstream message getting converted and compressed

Since this function has to iterate through the message as it is performing the conversion it will have an execution time of $O(n)$, where n is the number of symbols in the message.

Theoretically, any of the symbols seen in a complete ASCII could be sent to the AutroLink v2. This means that there are $256 - 22 = 236$ invalid characters. There is no logical way of converting any other symbols than 0-9, A-F or a-f. A check is therefore performed as the function iterates through the `hex_str_t`, to see if any other symbols than the valid ones are used.

The longest possible valid AL_Com+ message is 17 bytes long, as stated in Section 5.3.1. Checks could be implemented in both the upstream and downstream conversion functions in order to detect messages that are longer than 17 bytes and 34 bytes respectively. Failing the checks could either crash the system or simply refrain from conveying the message to the module stack. However, sending and receiving messages that are too long might be useful during debugging in order to detect or reproduce erroneous behavior. The checks for message length were therefore not implemented.

Upstream conversion

The function for the conversion of the upstream messages need to do the opposite of the downstream conversion function, converting from hexadecimal values to readable symbols. As input, the function takes the upstream message in the form of a `bin_msg_t` struct. The first and second nibble of each byte of the input array needs to be analyzed, and it's contents converted to readable symbols. This was

done using a for-loop that iterated through the array twice. On the even-numbered iterations it would look at the first nibble of the current byte, and on odd-numbered iterations it would look at the second nibble. As the array gets analyzed the information gets stored in a new array twice the length of the input array. If the current element had a value between 0 and 9, 48 gets added. With a value between 10 and 15, 55 gets added. Since this function iterates through the message twice it has an execution time of $O(2n)$. When keeping the largest factor of the expression we are left with $O(n)$.

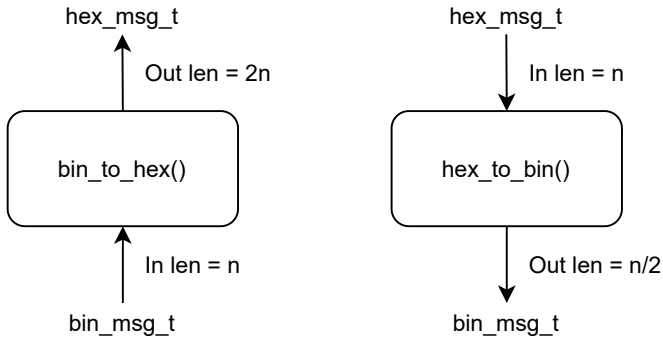

Figure 5.8: Visualization of the two functions that convert between readable symbols and hexadecimals

5.4.3 AL_Com and AL_Com+ conversion

As stated in Section 4.2, the AutroLink v2 should be able to convert AL_Com+ directives to AL_Com directives and vice versa. Following the information gathered in Section 2.6, we see that the AL_Com+ directives are similar to the AL_Com directives, save for one extra header byte and a different checksum.

For the upstream messages, the conversion entails removing the header byte and recalculating the checksum. The details of the checksum calculation is regarded as proprietary information, and will thus not be explained. The downstream messages need to be padded with a header byte that signifies it is a directive along with the address of a module. This poses a small problem. The AL_Com message sent from the client does not contain any information about which module in the module stack it should be sent to. One way to handle this is to have two topics per module stack unit, where one topic is meant for the upstream messages from that module, while the other is for the downstream messages to that module. However, during this project this was not implemented. The messages were rather consequently sent to the address of the loop driver in the testing setup.

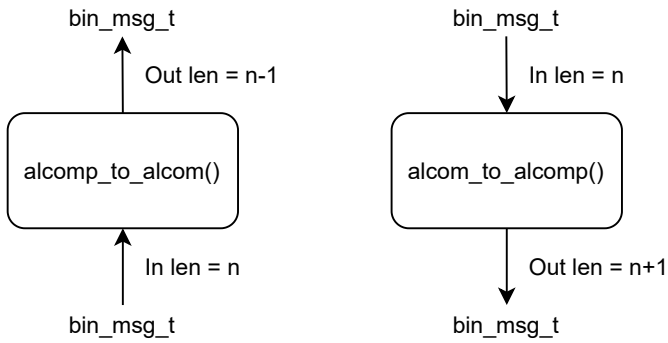

Figure 5.9: Visualization of the two functions that convert between AL_Com and AL_Com+

Two functions were made, one for the upstream messages called `alcomp_to_alcom()` and one for the downstream messages called `alcom_to_alcomp()`. These are illustrated in Figure 5.9. For these algorithms, the length of the output array is known as it is part of the input structure. Checks were therefore implemented to see if the output length matched the expectations.

5.4.4 UART handling

Similar to the implementation in the specialization project [1], the UART handling was set up using with an Interrupt Service Routine (ISR) using Zephyr's UART library [37]. In order to map the UART device's signal lines to the nRF9160 DK's GPIO a `.overlay` boardfile was created, specifying which GPIO pins were to be used for the UART communication and the baud rate of the communication. Since the module stack communicates at a signal rate of 9600 baud, this was reflected in the boardfile. This file is located at the root of the project folder and its contents can be seen below.

```
&uart0 {
        current-speed = <9600>;
        status = "ok";
        tx-pin = <29>;
        rx-pin = <28>;
};
```

Zephyr's UART library has a function for manipulating the line control called `uart_line_ctrl_set`. This function can be used to control the RTS signal of the UART. However, due to some unexpected behavior when using this function it was opted to seek another solution. Two GPIO pins were used for the RTS and CTS signal lines instead. These were controlled using Zephyr's standard GPIO library functions. A function called `gpio_pin_write()` can set the state of GPIOs defined

as outputs either high or low. Another function, called `gpio_pin_read()` can read the state of GPIO defined as inputs [37].

Sending UART messages

A function to send the AL_Com+ messages to the AutroSafe system was written. It was modeled after the flow chart in Figure 5.10. This function takes the `bin_msg_t` structure as input. Zephyr's GPIO library function `uart_poll_out()` was then used to write the message byte for byte to the UART TX pin. This iterates through the message using a for-loop and the information about the message length from the input structure. This function makes up the "UART send" block in Figure 5.4.

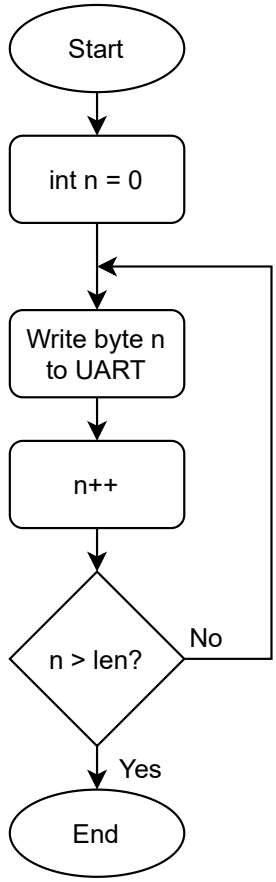

Figure 5.10: Flow chart depicting the process of transmitting a message byte for byte through UART

Reading UART messages

The function that read the incoming AL_Com+ messages from UART from the specialization project code differs from the method employed here. The AutroLink v1 instead analyzed the information in the AL_Com+ directives to determine the length of the current message, and also checked the checksum. The reason this function was not reused is because the nature of this approach may make it difficult to detect erroneous behavior. Instead, it was implemented based on information about the AL_Com+ protocol's timing restrictions.

In order to read the messages that have been picked up by the UART ISR, it is possible to wait on the FIFO queue from the ISR. This was implemented using a do-while-loop, which ensures that the thread waits on the queue at least once. Once the thread is waiting on the queue, one of two things can happen. Either a byte from the message gets picked up or the waiting expires due to a timeout. When waiting for the first byte of a response from the module stack, the function waits on the ISR FIFO queue for 100 ms. If a byte of data arrives before or within this time, the timeout timer is set to 10 ms. This is done in order to not pick up two AL_Com+ messages being sent after one another. Looking back at Section 2.6, the AL_Com+ transmission rate is 9600 baud. This gives the individual bits in the transmission a time of $\frac{1}{9600} \approx 1.04^{-4}s$, 104 microseconds between them. Consequently, one byte of data requires roughly 824 microseconds. Since waiting for 10 ms is longer than the time between bytes, yet shorter than the shortest retransmission time of 33 ms, it is deemed appropriate. Once waiting on the FIFO times out, information the byte array along with the message length are returned in the form of a pointer to a `bin_msg_t` structure. This function along with the ISR makes up the "UART read" block in Figure 5.4.

5.5 Main file

The main file is a necessary part of any C-based program. However, since all of the prototype's functionality has been modularized, written their own .c and .h files, and designed for a thread implementation, the responsibilities of the main loop are minimal. At startup it sets the values of the global variables that control whether the AutroLink v2 should process the downstream and upstream messages as AL_Com or AL_Com+. By default this is set to AL_Com+. The two threads, the MQTT client and the AutroSafe interface module, were declared at compile time using the macro `K_THREAD_DEFINE` [38]. Beyond this the main file has no other obligations.

5.6 Scheduling and priority

In order to schedule the different tasks, flat priority was used between the MQTT client module and the AutroSafe interface module, due to them both yielding often. The ISRs are given a higher priority than the aforementioned modules, while the system workqueue thread has a lower priority.

The scheduling strategy that was used to handle the prototype's tasks is called "longest wait first", which chooses the task in the queue that has been waiting the longest. This can also be viewed as a FIFO queue. This scheduling strategy has no preemption, meaning that tasks are not put on hold in favor of a higher priority task that appears. This means that a higher priority task may have to wait for a lower priority task to finish. This is, however, not deemed as a problem since most of the system's tasks are relatively short. The end goal here is to transmit the upstream messages in a timely manner, meaning that the processing and processing delay introduced by the prototype should be as small as possible. With this setup, there is a possibility for starvation to occur. This is however only if either the module stack were to continuously transmit AL_Com+ messages, or either of the software modules halting. These are situations that are not meant to happen, nor are they likely to happen. This implementation is therefore deemed adequate.

5.7 Firmware structure

The finished firmware's memory requirements are shown in Table 5.2. These numbers are provided when flashing the application using West. They also show that there is more space left in both SRAM and flash memory if changes or additions to the prototype's firmware is necessary in the future.

Memory region	Used Size	Region Size	%age Used
FLASH:	88712 B	976 KB	8.88%
SRAM:	43996 B	128 KB	33.57%

Table 5.2: The AutroLink v2 memory after flashing the latest firmware

In order for the modules to interact with each other, certain functionality had to be inherited. The main file needed to be able to call the main loops from both the MQTT client module and the AutroSafe interface module module in order to spawn their threads. Additionally, in order to access the downstream FIFO queue and the function for adding tasks to the workqueue, the AutroSafe interface module . The necessary functionality was made available from each module in their respective header file, which can be seen as the files marked with ".h" in Figure 5.11. The dotted lines show which header file belongs to which source file. The arrows show which header files are included in the source files.

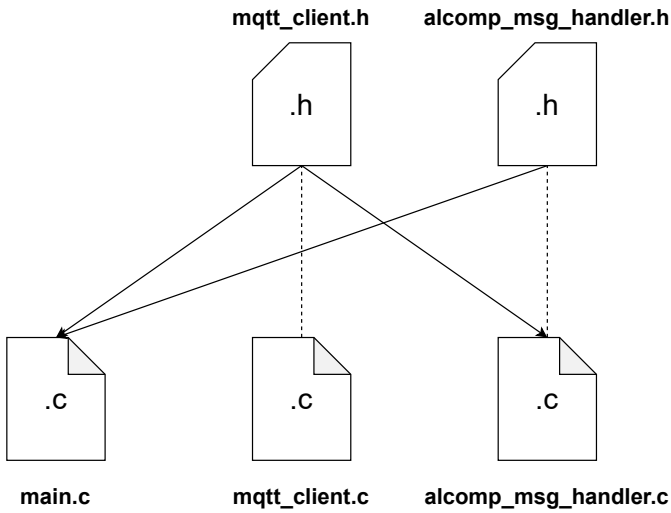

Figure 5.11: The AutroLink v2's code separation into different files with corresponding headers. The arrows show which file includes which header.

These header files were given include guards, since the main file includes the mqtt message handler's header file twice. Once directly through inclusion and once indirectly by including the header file of the AutroSafe interface module.

5.7.1 Module startup sequence

As explained in Section 5.5, the main file spawns the MQTT client thread and the AutroSafe interface module thread. Beyond this, it is necessary to ensure that some initialization and routines run before others. As the prototype is powered up, logically, one of the threads are bound to finish its initialization before the other. There are two possible scenarios of partial functionality before the two main modules are ready. The MQTT client can start first, in which case the operator is able to communicate with the AutroLink prototype, but not the AutroSafe system. Regardless, the prototype will not be ready for use before all modules have started up, so this argument is moot. Instead, it was opted to look at a startup sequence that ensured could ensure stability.

The MQTT client thread is the one that spawns the system workqueue thread by initializing the work items. This should be done before the AutroSafe interface module thread is initialized, since calling the function to put information into a work item before it is initialized would cause a crash. In normal circumstances this would not happen, as the modules in the module stack only respond to messages and will not send messages of its own volition. The MQTT client also initializes the FIFO queue used to pass information to the AutroSafe interface module. Attempting to wait on an uninitialized queue would also cause a crash. On the other hand, passing a downstream message into the FIFO queue before the AutroSafe interface

module is initalized will not cause any problems. A semaphore is therefore used to let the MQTT client thread perform its initialization first. Before the AutroSafe interface module performs its initialization it requests the init-semaphore. This semaphore is initialized witha value of zero, making the AutroSafe interface thread block. The MQTT client module will then be the only thread that is not blocked, and it will perform its initialization. First the system workqueue thread will be started, then the MQTT client will attempt to establish a connection with the MQTT server. As the MQTT event handler receives a successfull `CONNACK` message upon achieving connection to the MQTT server, it will release the semaphore to the AutroSafe interface module, allowing it to continue execution. This was implemented in a way that ensured it would not end up in a deadlock.

Chapter 6

Tests and results

The testing methods used are inspired by IEEE829, with separation into unit, integration and system testing. The purpose of splitting up the testing this way is to be able to perform and document the tests in an orderly manner. It will also give a clear indication if any submodule or module is unable to meet its specifications. Additionally, the higher-level tests may overlap multiple of the specifications.

In order to preserve Autronica's intellectual property, no actual input or output data will be described in detail in this section, only the general setup of the tests. As a consequence of this, information from the unit testing in Section 6.1 and the integration testing in Section 6.2 will be limited. The system tests are however more detailed. If a test uncovered erroneous behavior, the appropriate code was corrected before running the test again. The tests documented in this chapter are the final tests and results.

6.1 Unit testing

In this testing level, individual hardware and software components were tested. The tests were chosen based on engineering judgment based on the source code and knowledge of the protocol.

The tests that were performed were a mix of automated testing and user driven testing. This was because of the dependency on hardware I/O on a lot of the software. No time was invested into creating a simulator as this was out of scope.

6.1.1 Automated testing of software components

The conversion functions from Section 5.4.2 were tested using automated testing. The method used was black box testing, where the tests where designed from the specifications of the functions to be tested. This is depicted as a flow chart in Figure 6.1. After manually calculating the expected output based on a valid input, the valid input is fed into the corresponding function. If the function produces an output, this is compared to the expected output. If they are equal, the function is performing as expected by the specification.

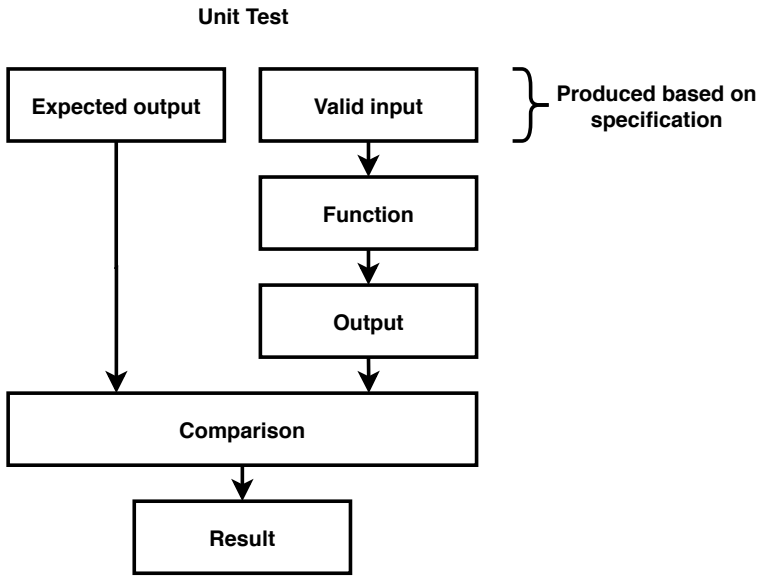

Figure 6.1: Flow chart of the unit testing procedure

Several valid inputs were tested on each corresponding function. However, these tests only uncovered if the functions produced the correct output. It was also necessary to test if the functions failed correctly. Therefore, both fail criteria and pass criteria were defined for these tests.

Pass criteria:

- The function produces the expected output given a valid input

Fail criteria:

- The function produces unexpected output given valid input
- The function does not stop execution when receiving invalid input
- The function stops execution when receiving valid input

The first and last fail criteria were triggered a few times during the early stages of development. Information from these were used to rectify the functions. In order to test the second fail criteria, the functions were given inputs that either based on the specifications or based on engineering knowledge of the programming would be invalid. The `hex_to_bin()` function had some additional tests performed on it. One of the tests were to set the length field of the input structure to an odd number. Another test used invalid characters in the input array.

Results

The four conversion functions passed the tests that were conducted, which signifies that they their required functionality was achieved.

6.1.2 Manual testing of hardware specific functions

The AutroLink v2's UART reading and writing capabilities were tested. Instead of connecting with the AutroSafe system, the prototype was connected to the development computer. The equipment necessary for these tests were the AutroLink v2 prototype, a RS-232 to Universal Serial Bus (USB) cable and a computer with the PuTTY software. The AutroLink v2's debug output was connected with the development computer using a micro-usb cable.

Reading UART messages

To test the prototype's ability to read UART at 9600 baud, PuTTy was set up with these attributes. A small script was written for the prototype that would echo the received UART messages to the debug output. The Powershell script was used to read this output. When using PuTTY to send messages to the prototype, these were echoed in the debug output. The results of this test showed that the prototype was able to read UART at 9600 baud.

Writing UART messages

To test the prototype's ability to transmit UART messages at 9600 baud, a script was written that caused the prototype to send a UART message every 5 seconds. This time interval was chosen arbitrarily. Here, PuTTy was used to read the messages that the prototype sent. The AutroLink v2 performed this task as expected, being able to send messages over UART at a symbol rate of 9600 baud.

6.2 Integration testing

At this testing level, groups of related units are tested together. The goal of the tests are to verify that the units interact with each other correctly.

6.2.1 Message decoder

The message decoder submodule of the AutroSafe interface module was tested. This module consists of the two downstream conversion functions from Section 5.4.2, which can be seen to the left in Figure 6.2. For this test, a script was written in order to input a AL_Com directive consisting of readable symbols into the `hex_to_bin()` function, which then passed it's output to the function that converts AL_Com directives to AL_Com+ directives. This signifies that this submodule was able to meet specification 4.

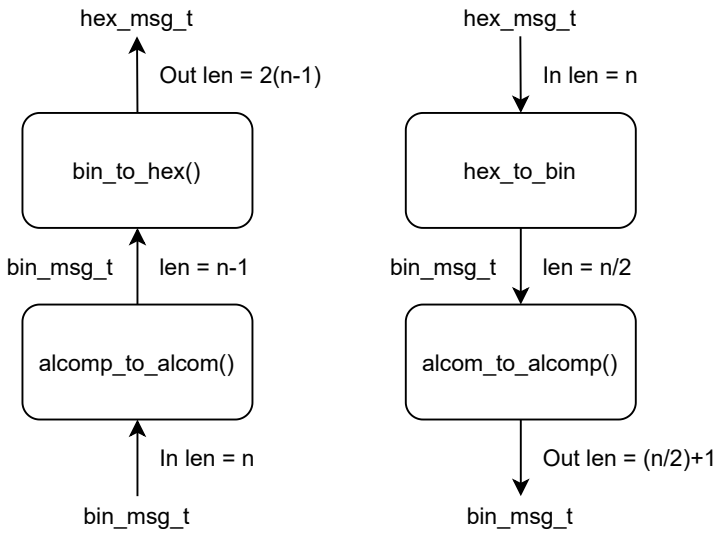

Figure 6.2: From left to right: Message encoder, message decoder

6.2.2 Message encoder

This submodule was tested much in the same manner as the message decoder submodule. An AL_Com+ directive was fed into the upstream directive converter, which then fed it's output into the `bin_to_hex()` conversion function, converting the hexadecimals to readable symbols. This was written in a script, much like in the message decoder test. This signifies that this submodule was able to meet specification 4.

6.2.3 MQTT event handler

When testing the MQTT event handler, a version of the firmware where only the MQTT client module's thread was started was flashed onto the development kit. Changes were also made so that the MQTT event handler would not attempt to pass information to the AutroSafe interface module, as its thread was not running. Instead it would publish a MQTT message to the upstream topic upon receiving something on the downstream topic. After initialization, the MQTT client would attempt to connect to the specified broker and subscribe to the specified topic. The module succeeded in doing this, meaning that it met specification 1: Connecting and communicating with the specified MQTT server. From the AutroLink v2's debug output it was possible to see the MQTT client's interactions with the MQTT server.

Two additional MQTT clients were set up using Mosquitto [8]. One of the clients subscribed to the topic for upstream messages, while the other was ready to publish a message to the downstream topic. After the AutroLink v2 had connected with

the MQTT server, the second Mosquitto client sent a message via the downstream topic, causing the MQTT event handler to respond. The debug output showed the MQTT traffic, which can be seen in Listing 6.1. The configuration topic was also tested in a similar manner. This signifies that the MQTT event handler was able to meet the requirements in specification 2 and 3.

```
1 [mqtt_evt_handler:207] MQTT CONNACK 0
2 Subscribing to: ALK_v2/panel-1/Downstream
3 [mqtt_evt_handler:315] SUBACK packet id: 1234
4 [mqtt_evt_handler:226] MQTT PUBLISH
5 Received: XX
6 Publishing: YY
7 to topic: ALK_v2/panel-1/Upstream
8 [mqtt_evt_handler:305] PUBACK packet id: 61043
```

Listing 6.1: MQTT traffic from the AutroLink v2's debug output

6.2.4 MQTT publisher module

In order to test that the workqueue implementation of the MQTT publisher module functioned according to the specification, a message publishing test was set up. A script was written, that was executed after the MQTT client module had established a connection to the MQTT broker and the workqueue items had been initialized. The script was hardcoded to put an item into the workqueue every 100 ms. This number was chosen arbitrarily.

The same client as in the previous test was still subscribed to the upstream topic. Here, it was observed that the messages arrived, meaning that the MQTT publisher submodule also met the requirement in specification 2.

6.3 System testing

After conducting the integration tests, the next level of verifications are the system tests. These tests are conducted on the complete, integrated system in order to evaluate the system's compliance with its specified requirements.

The following tests all had the same test setup and equipment. The testing equipment is listed below.

- AutroLink v2 prototype
- DB9 to 10-pin latch contact cable
- BSS-310A Power supply module [39]
- BSL-310 Communication module [40]
- BSD-310 Loop driver module [3]
- BPS-410 Power supply unit [41]
- BHH-300 Optical smoke detector [42]

The test setup consisted of a module stack, a detection loop and a power supply. The module stack contained the minimum modules in order to have a functioning detection loop, with a power supply module, a communication module and a single loop driver module. The detection loop contained a single loop unit, which was an optical smoke detector. The AutroLink v2 prototype could connect with this test setup using the DB9 to 10-pin latch contact cable, which is depicted in Figure 2.10. The prototype's debug output is connected to a computer running the Powershell script mentioned in Section 5.1.

6.3.1 AutroLink v2 startup test

The first system test that was performed on the AutroLink v2 was the startup test. In this test, the AutroLink v2 was connected to the communication module of the module stack with the DB9 to 10-pin latch contact cable as seen in Figure 2.10. Additionally, the AutroLink v2's debug output was connected to a computer running the Powershell script mentioned in Section 5.1.

First the module stack was supplied with power. Then the AutroLink v2 was turned on. Here it was possible to read from the debug output that the prototype was running its startup procedures. It connected with the MQTT server and subscribed to the specified topic. Afterwards it started communicating with the module stack. As it started the module stack addressing procedure it enabled the RTS signal line. This was indicated by the LEDs of the communication module visibly lighting up. The prototype then started sending ENTX flow control messages to the module stack and receiving responses from the modules. Once the modules received an ENTX along with their assigned address, they either responded with a NOP flow control message, or a directive. This was observed both through the debug output as well as visibly observed by the LEDs of the modules turning on one after another.

In order to test that it now was possible to communicate with the individual modules, messages were sent from the Mosquitto MQTT client running on the computer. When sending an ENTX to the individual modules, they either responded with a NOP flow control message or a DIR. The upstream and downstream messages were visible both in the debug output as well as appearing as messages in the Mosquitto MQTT client. This concluded the AutroLink v2 startup test. The results show that the prototype was able to communicate with the module stack, control the RTS signal line, and assign unique addresses to the modules. It was also able to receive MQTT messages from the Mosquitto MQTT client through the MQTT server, convert the downstream messages and convey them to the module stack. It was able to pick up responses from the module stack, convert them and send them to the Mosquitto MQTT client.

6.3.2 Loop raising test

The purpose of this test is to see if it is possible to perform specific procedures on the module stack and connected detection loop. The loop raising procedure is one such procedure that will be performed both during service and during commissioning.

This test was performed after the AutroLink v2 startup test. At this point, the AutroLink v2 had already assigned addresses to the modules in the module stack, and the communication between the Mosquitto MQTT clients and the modules in the module stack had already been verified.

This test was performed by manually sending the necessary directives and flow control messages from the Mosquitto MQTT client in order to perform the loop raising procedure explained in Section 2.8.4.

The results from this test shows that it is possible to manually perform procedures on the module stack and connected detection loops. Doing it manually let the operator observe each response as messages got sent.

6.3.3 Detection unit parameter change test

With the detection loop raised after performing the previous test, it is possible to test that routines can be performed on a raised loop. These procedures involve communicating with the individual loop units on the detection loop. One way to test the communication is to perform a parameter change. This test was set up to change the state of the BHH-300 optical smoke detector's LED by sending directives requesting the change of specific parameters. This is how it is also done when using the AS-2000 software to perform the LED test procedure, described in Section 2.8.1.

Directives were sent from the Mosquitto MQTT client instructing the detector to switch it's LED on and off. The parameter change was visually confirmed as the state of the detector's LED was changed.

6.3.4 AL_Com+ abstraction test

In order to test the prototype's ability to function as a transparent link between Autronica's prototype software, mentioned in Section 4.2, and the physical detection loop, a test was set up in collaboration with an engineer from Autronica's R&D department. Unfortunately, as this software was still under development, the integration towards AutroMaster ISEMS was not completed in time of this test.

The software was set to mimic a panel and loop driver, and would perform the loop raise procedure automatically. The software sent AL_Com directives consisting of readable symbols, which the AutroLink v2 then converted to hexadecimal values before converting the directive to AL_Com+. Before the test started, the software and the AutroLink v2 prototype was connected to the same MQTT server, and the AutroLink v2 had performed the module stack addressing procedure.

The software and the prototype was able to communicate with each other through the MQTT server, successfully raising the loop. This test verified that the AutroLink v2 could be used as a transparent link form the testing software to a physical detection loop. It also showed that the AutroLink v2 was able to convert AL_Com directives to AL_Com+ and vice versa.

This test did uncover an error done in the programming. One of the directives used in the loop raising procedure made the loop driver respond with an ACK flow control message, which made the AutroLink v2 expect a following directive that never came. This made the communication halt, causing the prototype to crash and restart. This error was quickly corrected. The rest of the test went as planned with no unexpected behavior.

6.4 Summary of results

The unit tests verified that individual functions did as planned. The integration tests showed that the individual functions and hardware modules were able to interact with each other. The system testing tied it all together and showed that the system as a whole was able to meet the specification listed in Section 4.6, and also the requirements in Section 4.4.

Chapter 7

Discussion

7.1 The tests

The testing performed on the AutroLink v2 prototype verifies that it was designed according to the specifications, including that the modules that were defined in Section 4.5 had the correct functionality. However, validating the requirements that were defined for the new prototype proved more challenging. In the end this is partially based subjective experiences working with the testing equipment, partially based on the documentation available on the service and commissioning routines, and partially based on feedback received through conversations with Autronica's employees. A service engineer was able to evaluate the requirements. They provided feedback stating that the proposed functionality, as well as the requirements seemed appropriate. They were unfortunately not able to attend the testing of the prototype. On the other hand, the abstraction test that is documented in Section 6.3.4, was conducted together with the engineer from Autronica's R&D department, and was deemed successful. This test encompassed many of the previous test's specifications, which by proxy gives a pointer towards the validation of the requirements.

Another point to address is the minimal test setup that was used when conducting the tests on the AutroLink v2 prototype. Although it was a fully functioning AutroSafe setup, it would be interesting to test the AutroLink prototype on a full scale industrial installation. This could be done in combination with a commissioning project. By doing this, valuable information about how the AutroLink v2 would have been used could have been gathered. In addition to the large scale testing, tests should also be performed in different environments. The tests were performed in the author's one room student housing condo in Trondheim, where cellular coverage was no issue. This is not a downside, as having a relatively controlled environment during the initial testing Some of Autronica's installations are located ships, floating rigs and oil rigs. At these locations, cellular coverage may vary greatly.

Normally, both the tests and the code written for a project as peer reviewed at Autronica. The main reason for this is that AutroSafe system is safety critical. This should also have applied to both the testing done in this thesis, as well as the firmware code written for the AutroLink v2 prototype. This was unfortunately not possible during the duration of this thesis.

7.2 The prototype

This section contains discussions about the prototype, the functionality, the development and the implementation.

Working with the NCS release tag v1.2.0

When starting the practical part of this master's thesis, namely coding the firmware for the new prototype, it was decided to work on the same tag of NCS as was used in the specialization project. This decision was made based on the safety of knowing that the firmware from the specialization project worked just fine on this tag. Additionally, it was motivated by the fact that moving the project to a new tag would incur additional work. At the time, it seemed like the best choice, but later on in the development it did have certain consequences. The most prevalent issue was the library functions to use the RTS signal line of the UART. This problem was circumvented by not using the functions, and rather use functions from Zephyr's GPIO library to manually control the RTS signal. As both NCS and the Zephyr project are still under development, using an outdated tag could introduce issues with stability and functionality. These issues were, however, not experienced.

Wireless connection

Tying back to Section 7.1, where the topic of cellular coverage was brought up. One of the reasons the LTE-M was chosen as the way to connect with the Internet is the fact that this network provides good coverage over Norway's landmass. Another reason is that this IoT oriented network in particular supports roaming, something that was deemed necessary for moving AutroSafe installations, such as the ones on ships. However, such environments have a tendency to not be especially penetrable with regards to signals, as the hull and sections of the ship have a tendency to be made out of thick steel. A limitation that appears because of this is that the AutroLink v2, and AutroLink v1 for that matter, require stable cellular coverage. It may therefore not be fit for use on ships, in basements, or industrial buildings for that matter. Therefore, for the prototype to be useful in these situations, alternative connectivity methods to LTE-M should be explored. This project is able to display the prototype's proof of concept regardless. Additionally, the nRF9160's modem firmware has different certifications depending on the version. The one used during this thesis is certified for use in Norway, as most version of the firmware are. It is, however, important to note that some versions are not supported in different regions of the world.

Automatic procedures

The prototype was able to perform the module stack addressing automatically at startup. The main reason for this arose when Autronica requested the possibility of communicating with the prototype using AL_Com. It was also practical when

using AL_Com+ to communicate, especially in terms of being able to start communicating straight away. It could instead have been implemented in a way where the operator wold have to instruct the AutroLink v2 to perform the module stack addressing. As such, this procedure could have been implemented as a function that the operator could call via the configuration topic or a similarly defined topic for these kinds of tasks. Other routines and procedures such as loop raising could also have been implemented so that they could be run automatically or by the operators request.

Security concerns

The AutroLink v1 had some inherent security features in that it was connected to a port that only allowed reading of the AL_Com+ messages. As such, the introduction of this prototype into an AutroSafe system did not put the system at any risk. The AutroLink v2 on the other hand, connects directly into a module stack's communication module, and reads and writes messages at the operator's will. When such a prototype is introduced into a system, the prototype should have built in security features that ensure the safety of the connected system, as stated by the SeSa method [12]. For the AutroLink v2, this would entail having knowledge of the valid directives and flow control messages, as well as when each of them are safe to use. As such, the prototype could even protect the system against human error from the operator. It would, however, have been a more difficult job to implement such functionality compared to what was achieved during this project. As part of the literature study, security in MQTT was studied, such as the use of TLS and mutual authentication. This could help protect a tool such as the AutroLink v2 from malicious cybersecurity attacks. Implementing this was however not prioritized in this thesis, and would not have been used during the testing conducted in collaboration with Autronica either way due to their prototype software not supporting it, as stated in Section 4.2. Instead, this prototype stands more like a proof of concept for the idea of introducing remote connections to a AutroSafe system, with the intention of using it to aid in service and commissioning. and has it's intended use limited to during service and commissioning situations. This is somewhat supported by the information in DNVGL's report RP-108 [13], where it is stated that temporarily connected equipment do not have to abide by the same strict requirements as equipment that is connected permanently.

The utility of this prototype

Future fire detection systems from Autronica are likely offer remote service inherently. This raises the question if there is much utility in a solution like the AutroLink v2. Another viewpoint is that Autronica should be able to provide remote services to customers of the old fire detection systems as time passes. In this case, a prototype like the AutroLink v2 may be a more viable option.

7.3 Achievement of goals

In this section, it will be discussed if the tasks presented in Section 1.2 were achieved to an adequate degree. Task 5 is omitted from this discussion as it pertains the discussion covered in Section 7.1 and Section 7.2, and to an extent this chapter as a whole. Task 6 is also omitted as it is covered by Section 8.2.

Task 1 - Equipment research

The first task was to get familiar with and describe the relevant equipment from Autronica, as well as the service and commissioning routines. The work to achieve this task resulted in Chapter 2, the system description. Documentation from Autronica, both publicly available and internal, was used to get an overview of the AutroSafe system. Their proprietary information was used to a greater extent when going into detail about the protocols, module stack and detection loops. The same applies for the sections written about their tools and service routines. The relevant parts of the system and the routines were researched and documented to such a degree that it was possible to define new requirements for the AutroLink v2 prototype. This task is therefore deemed successful.

A challenge that presented itself when working on this task was that much of the necessary information regarding Autronica's components, routines and the AutroSafe system is proprietary. It was therefore necessary to strike a balance between describing the system in enough detail for the readers of this rapport, while not leaking any of Autronica's proprietary information. However, in spite of this challenge, the system was described in broad strokes, with protected information abstracted to give a description of purpose/functionality without describing the inner workings. If this report was meant for internal use within Autronica, the descriptions, samples and illustrations could have been more detailed. This would be more valuable for Autronica.

Task 2 - Literature study

The second task was to conduct a literature study on relevant subjects, as well as to examine whether commissioning and service with remote access had been discussed in literature. As stated in Section 1.3, specific search terms were used to try and locate reports and articles of relevance through search engines such as Google Scholar and NTNU Oria. Here, many of the search results were articles from tech magazines describing up-and-coming products and solutions for remote service and streamlining thereof. These results did not go into a meaningful degree of technical detail. The rest of the results were standards, technical reports and research documents pertaining remote access to safety critical systems for the oil and gas industry. These search results mainly had a great focus on cybersecurity, which was outside the scope definition of this thesis. A few sources from each end of this spectrum was chosen and documented as part of Chapter 3. Information regarding the other subjects of the literature study were readily available. Although

there still is a wish to find more relevant sources to the subjects, this task is deemed completed to an adequate degree.

Task 3 - Prototype

The third task was to propose a design for the prototype, perform a structured analysis and present specifications. The structured analysis and presentation of the specifications were performed as part of Chapter 4. The proposed design was presented in Section 5.2. The work performed during this thesis mainly focused on the functionality and firmware of the prototype. The hardware of the prototype is a minimal implementation that facilitates the defined functionality. This task is therefore deemed completed.

It also means that since a lot of the ground work has been done and documented, reworking the specifications won't necessarily have to be done from the ground up. And again, as stated in Section 7.1, performing an acceptance test in a large scale system could help point out weakly defined requirements and specifications.

Task 4 - Testing to design

The fourth task was to implement and test the proposed prototype design. The implementation is documented throughout Chapter 5 while the testing is documented in Chapter 6. Both of these tasks were completed to an adequate degree as the prototype has been documented and met its specification through testing.

Chapter 8

Conclusion and future work

In this final chapter of the thesis, the findings and results are concluded. Certain opportunities and tasks outside the scope of the thesis are presented as proposals for future work.

8.1 Conclusions

In this project we were able to verify that the AutroLink prototype could be useful as a tool for service and commissioning. This was done by introducing new functionality that was located through an analysis of the AutroSafe system and Autronica's service and commissioning routines.

By connecting directly with the communication module in the module stack the prototype can perform certain routines automatically, while it facilitates others through two-way communication. Service engineers and other trained personnel can connect to the prototype wirelessly in order to gain access to communication that reaches the units in the module stack, as well as the units of the detection loops. This can benefit the service and commissioning routines as it may reduce the number of people necessary on site to perform certain routines, as well as simplifying the process of providing expert competence from a remote location.

8.2 Future work

The following subjects were located throughout the work on this thesis. They were either outside the scope of the thesis, or were alternatives to methods used. They are presented as possibilities for future work.

Further testing

Due to a limited test setup and restrictions due to the Covid-19 pandemic, the testing performed on the AutroLink v2 prototype was limited. The prototype should receive further testing by trained personnel and service engineers in a large scale system. This could help determine the usefulness of the AutroLink v2 prototype in remote service situations, as well as during parts of the commissioning of the AutroSafe system.

Message translation

The messages were converted from hexadecimals to ASCII encoded symbols for readability purposes. However, to most service engineers and trained personnel, these messages are hard to read unless they are fluent in AL_Com. Using information about the directives, the messages could translated in a manner that better convey their meaning, making them inherently "human-readable".

Extended automatic functionality

The module stack addressing procedure was implemented to be run automatically. This was to expedite the service so that if the operator was to communicate using AL_Com instead of AL_Com+, it could be done right away. Other routines could also be implemented to run automatically. Alternatively, other routines could be made into functions that can be called by sending the AutroLink v2 specific configuration messages.

AS-2000 compatibility

The AS-2000 software is used throughout many of the service procedures and routines that are employed on the AutroSafe system today. By augmenting this software with an MQTT client it could be used remotely through the AutroLink v2 prototype. This could prove useful as trained personnel and service engineers already are familiar with the GUI of the software.

Alternatives to LTE-M

The AutroLink v2 relies on cellular coverage in order to be able to connect with the Internet connected MQTT server. More specifically, it relies on LTE-M coverage. There is, however, not necessarily coverage at every location Autronica has an AutroSafe installation. Examples of this are ships at sea, floating rigs and oil rigs. Alternative wireless communication methods should be explored for testing on such locations. The nRF9160 DK supports NB-IoT, which could be a starting point for this task, considering no hardware changes are necessary in order to test it.

Custom PCB

The prototype was conceptualized, implemented and tested on a nRF9160 DK from Nordic Semiconductor. For future work and testing, a custom PCB should be designed for the prototype.

Temporary fire detection

Sometimes, the AutroSafe fire detection system gets installed in onshore, offshore and marine vessels while the facilities are still under construction. Tools that are a fire-hazard in and of themselves, such as blow torches, welding-apparatus and angle grinders are often used in these situations. Here, temporary fire detection could

prove useful. A literature study could be conducted on systems for temporary fire detection. Furthermore, it would be interesting to see if a basic cause and effect algorithm could be implemented on the AutroLink prototype for testing purposes. This way, the AutroLink could function as a temporary controller for the module stack and detection loop, and thus provide rudimentary fire detection.

References

[1] Håkon F. Johansen. "Development of a remotely operated equipment for monitoring internal bus communication in the Autrosafe 4 Fire Detection System." Specialization project report TTK4551. Høgskoleringen 1, 7491 Trondheim: Norwegian University of Science and Technology, June 2020.

[2] *AutroSafe 4 System Description*. eng. 2017. URL: https://product.autronicafire.com/fileshare/fileupload/11868/asafesystemd_egb_20.pdf.

[3] *Loop driver module BSD-310/311 product datasheet*. eng. 2014. URL: http://product.autronicafire.com/fileshare/filArkivRoot/produkt/dokumentasjon/bsd310_cgb.pdf.

[4] Autronica [Internal document]. *Loop handling - high level description*. eng. 2020.

[5] Autronica [Internal document]. *Basic Al_ Com description*. eng. 2012.

[6] *User Guide, Loop Diagnostic Tool AS2000*. eng. 2015. URL: https://product.autronicafire.com/fileshare/filArkivRoot/produkt/dokumentasjon/asafeas_fgb.pdf.

[7] *AutroSafe 4 Commissioning Handbook*. eng. 2014. URL: https://product.autronicafire.com/fileshare/fileupload/14316/asafecommiss_egb.pdf.

[8] Roger Light. "Mosquitto: server and client implementation of the MQTT protocol." In: *Journal of Open Source Software* 2.13 (2017), p. 265. URL: https://doi.org/10.21105/joss.00265.

[9] Riccardo Masoni et al. "Supporting Remote Maintenance in Industry 4.0 through Augmented Reality." In: *Procedia Manufacturing* 11 (December 2017), pp. 1296–1302.

[10] *Remote Commissioning Possible, During COVID-19 Lockdown*. eng. August 2020. URL: https://www.schenckprocess.com/press-releases/Remote-Commissioning.

[11] *Remote Commissioning and Modernization: Coperion's Solution for Extruder Service in Challenging Times*. eng. August 2020. URL: https://www.coperion.com/en/news-media/newsroom/2020/remote-commissioning.

[12] Tor Olav Grøtan et al. *The SeSa Method for Assessing Secure Remote Access to Safety Instrumented Systems*. 2007. URL: https://www.sintef.no/globalassets/project/pds/reports/sintef-a1626-the-sesa-method-for-assessing-secure-remote-access-to-safety-instrumented-systems.pdf.

[13] *Cyber security in the oil and gas industry based on IEC 62443*. eng. 2017. URL: http://rules.dnvgl.com/docs/pdf/dnvgl/rp/2017-09/dnvgl-rp-g108.pdf.

[14] *MQTT Version 5.0 documentation*. 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf (visited on 03/10/2020).

[15] *MQTT*. eng. 2021. URL: https://docs.zephyrproject.org/latest/reference/networking/mqtt.html.

[16] *Wikipedia Transport Layer Security*. https://en.wikipedia.org/wiki/Transport_Layer_Security. Accessed: 2020-10-20.
[17] *Wikipedia Man-in-the-middle attack*. https://en.wikipedia.org/wiki/Man-in-the-middle_attack. Accessed: 2020-10-23.
[18] Nordic Semiconductor. *nRF9160: Simple MQTT*. URL: https://github.com/nrfconnect/sdk-nrf/tree/master/samples/nrf9160/mqtt_simple (visited on 03/10/2020).
[19] *freeRTOS coreMQTT Demo (Mutual Authentication)*. https://www.freertos.org/mqtt/mutual-authentication-mqtt-example.html. Accessed: 202-12-19.
[20] Phillip A. Laplante. *Real-Time Systems Design and Analysis: An Engineer's Handbook*. IEEE Press, 1992.
[21] M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. USA: Prentice-Hall, Inc., 1990.
[22] Allen B. Downey. *The little book of semaphores*. eng. 2016. URL: http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf.
[23] *Introduction to algorithms*. eng. Cambridge, Mass, 2009.
[24] Valerie Henson. *Crash-only software: More than meets the eye*. Ed. by LWN.com. July 2006. URL: https://lwn.net/Articles/191059/.
[25] "IEEE Standard for Software and System Test Documentation." In: *IEEE Std 829-2008* (2008), pp. 1–150.
[26] *Introduction to Zephyr*. eng. 2020. URL: https://docs.zephyrproject.org/latest/introduction/index.html.
[27] *Zephyr Kernel APIs*. eng. 2018. URL: https://docs.zephyrproject.org/1.9.0/api/kernel_api.html.
[28] *West (Zephyr's meta-tool)*. 2020. URL: https://docs.zephyrproject.org/latest/guides/west/index.html (visited on 05/14/2020).
[29] *Github Features*. https://github.com/features. Accessed: 2021-01-20.
[30] *nRF Connect SDK*. URL: https://www.nordicsemi.com/Software-and-tools/Software/nRF-Connect-SDK (visited on 03/15/2020).
[31] PuTTY team. *PuTTY Documentation Page*. 2020. URL: https://www.chiark.greenend.org.uk/~sgtatham/putty/docs.html (visited on 11/11/2020).
[32] *nRF9160: Simple MQTT*. eng. 2020. URL: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/1.2.0/nrf/samples/nrf9160/mqtt_simple/README.html.
[33] *Fifos*. eng. 2018. URL: https://docs.zephyrproject.org/1.9.0/kernel/data_passing/fifos.html.
[34] *Workqueues*. eng. 2018. URL: https://docs.zephyrproject.org/1.9.0/kernel/threads/workqueues.html.
[35] *The system workqueue*. eng. 2020. URL: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/1.2.0/zephyr/reference/kernel/threads/system_threads.html.
[36] Colin et al. Robertson. *Typedef Declarations*. 2020. URL: https://docs.microsoft.com/en-us/cpp/c-language/typedef-declarations?view=msvc-160 (visited on 01/01/2021).

[37] *UART*. eng. 2019. URL: https://docs.zephyrproject.org/2.1.0/reference/peripherals/uart.html.

[38] *Zephyr Kernel Threads*. eng. 2020. URL: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/1.2.0/zephyr/reference/kernel/threads/index.html.

[39] *Power supply module BSS-310A product datasheet*. eng. 2014. URL: https://product.autronicafire.com/fileshare/filArkivRoot/produkt/dokumentasjon/bsl310_cgb.pdf.

[40] *Communication module BSL-310 product datasheet*. eng. 2011. URL: https://product.autronicafire.com/fileshare/filArkivRoot/produkt/dokumentasjon/bsl310_cgb.pdf.

[41] *Power supply unit BPS-410 product datasheet*. eng. 2016. URL: https://product.autronicafire.com/fileshare/filArkivRoot/produkt/dokumentasjon/bps410_cgb.pdf.

[42] *Optical smoke detector BHH-300 product datasheet*. eng. 2009. URL: https://product.autronicafire.com/fileshare/filArkivRoot/produkt/dokumentasjon/bh300_cgb.pdf.

Appendices

Appendix A

Contents of zip-file

The zip-file contains:

- The latest source code of the project
- The Powershell script used for reading serial data
- The report from the specialization project

Håkon Flisnes Johansen

NTNU
Norwegian University of
Science and Technology